BuiltSteady Engineering Guide

Contents

[Source Control 2](#_Toc321814487)

[Repositories 2](#_Toc321814488)

[Documentation 2](#_Toc321814489)

[Structure 2](#_Toc321814490)

[Source Code 4](#_Toc321814491)

[Structure 4](#_Toc321814492)

[Enlisting and Making Changes 5](#_Toc321814493)

[**Naming Conventions** 5](#_Toc321814494)

[Branching 5](#_Toc321814495)

[Commands 5](#_Toc321814496)

[Conventions 6](#_Toc321814497)

[Build Environment 7](#_Toc321814498)

[Build 7](#_Toc321814499)

[Windows 7](#_Toc321814500)

[Azure Project 7](#_Toc321814501)

[iOS 7](#_Toc321814502)

[Coding Conventions 8](#_Toc321814503)

[CSharp 8](#_Toc321814504)

[Namespace Declarations 8](#_Toc321814505)

[Using Statements 8](#_Toc321814506)

[JavaScript 8](#_Toc321814507)

[T-Sql 8](#_Toc321814508)

[Debugging 9](#_Toc321814509)

[WinPhone Client 9](#_Toc321814510)

[Deployment 10](#_Toc321814511)

[Domain Management 10](#_Toc321814512)

[Azure Accounts 10](#_Toc321814513)

[Deployment Conventions 10](#_Toc321814514)

[SQL Azure 10](#_Toc321814515)

[DAC 10](#_Toc321814516)

[Backup 10](#_Toc321814517)

[Generating Data Script 11](#_Toc321814518)

[Rename 11](#_Toc321814519)

[dotnet-Host 11](#_Toc321814520)

[iOS Device 12](#_Toc321814521)

[Test 13](#_Toc321814522)

# Source Control

## Repositories

All documentation and source code are stored in a GitHub repository.

Directions for setting up GitHub for Windows are located at: <http://help.github.com/win-set-up-git>

Directions for setting up GitHub for Mac OSX are located at: <http://help.github.com/mac-set-up-git>

## Documentation

Product documentation and corporate website are stored in a GitHub repository named builtsteady.

To enlist in the builtsteady repository, open a git bash command window and clone the repository to your local machine:

git clone git@github.com:ogazitt/builtsteady.git

### Structure

The builtsteady repository is divided into two primary directories; documents and corpweb.

The documents directory contains informational documentation for builtsteady corporation and the product. This includes legal documentation, marketing materials, presentations, videos, etc… This repository does not contain technical documentation, as that is stored in the zaplify repository with the source code. The documents tree is organized by area; legal, product, etc…

The corpweb directory contains the projects and source code for the corporate website.

## Source Code

The projects, source code, and technical documentation are stored in a GitHub repository named zaplify.

To enlist in the zaplify repository, open a git bash command window and clone the repository to your local machine:

git clone git@github.com:ogazitt/zaplify.git

### Structure

The zaplify repository is divided into four primary directories; source, test, tools, and documents.

The source directory contains the projects and code for the product. The source tree is organized by area: services, devices, packages, and shared. The packages directory contains external open-source code utilized by the product. The shared directory contains internally developed code utilized across areas of the product.

The test directory contains the projects and code for testing the product. The test tree is organized by area mirroring the source tree, with the addition of a clients directory for implementing test clients for various services.

The tools directory contains external and internal tools used to development, management, deployment, etc.. of the product.

The documents directory contains technical documentation pertaining to architecture, design, building and deploying the product, management of cloud services, test plans, etc...

## Enlisting and Making Changes

The following are the common git commands used for adding, removing, or updating files in a repository:

git add **filename** // add or update file in local changes

git add **directory** // add or update entire directory in local changes

git rm **filename** // remove file from local changes

git mv **filename** // rename file in local changes

git status // review local changes

git commit –m '**comment**' // commit local changes

git commit –am '**comment**' // add all modified files and commit local changes

git push // push committed changes to remote repository

Additonal resources on making changes in Git:

<http://book.git-scm.com/3_normal_workflow.html>

## **Naming Conventions**

There are a few naming conventions that should be followed when adding new directories or files to the repository. External tools and packages are exempt from these conventions.

* Directory names **should** be lower-case and **must not** contain spaces. This is to eliminate confusion when interacting with GitHub which is case-sensitive.
* The names of files that contain code **should not** contain spaces. (includes code in source, test, and tools)
* The names of files that contain code **should** use pascal-casing. ( MyCodeModule.cs, Web.config, etc. )
* The names of files in a documents directory **may** contain spaces.

## Branching

In order to avoid conflicts between developers and maintain code corresponding to running deployments, it is strongly advised that all code changes be made in a branch derived from the master branch. Changes to the master branch should almost always be a result of merging changes from another branch.

### Commands

The following are the common git commands used for creating, managing, and merging branches in a repository:

git branch // list all existing branches

git branch **branchname** // create a new branch (from current branch)

git checkout **branchname** // switch to another branch

git push origin **branchname** // push committed branch changes to repository

git pull origin **branchname** // pull branch changes from repository

git merge **branchname** // merge branch changes (into current branch)

git branch –d **branchname** // delete a branch after merging (from master)

git branch –D **branchname** // delete a branch without merging (from master)

Additonal resources on using branches:

<http://book.git-scm.com/3_basic_branching_and_merging.html>

<http://learn.github.com/p/branching.html>

<http://progit.org/book/ch3-0.html>

### Conventions

There will always exist two branches that correspond to the current Production and Staging deployments. (see Deployment for conventions). An individual developer should adhere to the following process when implementing a set of features or fixes:

1. Create a development branch off the current master branch (e.g. dev1)

(master) $git branch dev1

1. Make changes in development branch and test changes locally

(master) $git branch dev1

1. Deploy development branch to the corresponding Azure development account
2. Test and verify development deployment is working properly
3. Create a deployment branch from that development branch using the current date in the branch name (e.g. dev1\_mmddyy). This branch is used to maintain the exact code that is currently running in a deployment. If critical fixes need to be made to the deployment, this branch can be used. A deployment branch can be deleted when it is replaced by a newer deployment branch.

(dev1) $git push origin dev1

(dev1) $git branch dev1\_041012

(dev1) $git checkout dev1\_041012

(dev1\_041012) $git push origin dev1\_041012

1. Merge the original development branch (e.g. dev1) back into master branch

(master) $git pull

(master) $git merge dev1\_041012

(master) $git push

1. Merge the latest master branch into the development branch before beginning work on the next set of fixes or features.

(master) $git checkout dev1

(dev1) $git merge master

(dev1) $git push origin dev1

The master branch will be used as the integration branch for fixes and features being worked on in the development branches. It is recommended that at least once per week, a new staging branch is created, deployed, and tested to the staging deployment. This will catch integration issues as multiple development branches are merged into the master branch. In some cases, a developer may catch integration issues when merging the latest master branch back into their development branch. At this point, the developer must create a new staging branch, resolve the integration issues, and update the staging deployment. Merging of other development branches into the master branch will be suspended until the staging integration is complete and merged back into the master branch.

If a set of features requires integration prior to merging with the master branch, a development branch can be created and shared between multiple developers, pushing and pulling the other’s changes as the features are being developed.

Periodically, the current known good staging branch and deployment will be promoted to production. After testing and validating the staging deployment is stable, a new production branch will be created from the staging branch and deployed to the production Azure account. The new production branch should be named using the current date in the branch name (e.g. production\_ddmmyy)

# Build Environment

# Build

## Windows

The main project for building the services and Windows clients is a Visual Studio solution located in the source directory: /zaplify/source/zaplify.sln

### Azure Project

The Azure project is setup with multiple configuration files. By default, the project will use the ‘Local’ configuration which is configured to run in the DevFabric and use databases on the local machine. In addition, there are configurations for each of the standard Azure accounts: Omri (production), Steve (staging), Dev1, and Dev2. One can run the DevFabric against another deployment by opening the Azure project properties, selecting the Development tab, and then choosing the specific configuration to use.

## iOS

The main project for building the iOS client is a solution located in the devices directory: /zaplify/source/devices/ios/ios.sln

# Coding Conventions

## CSharp

### Namespace Declarations

The namespace declaration should be the outermost containment for the document, with using statements and class definitions within. The convention for the namespace for a given source file is:

namespace {company}.{product}.{project}.{folder}

### Using Statements

The using statements should be placed at the top of the source file just inside the namespace declaration. By convention the system namespace references are declared first in alphabetical order, followed by third-party namespace references in alphabetical order, followed by the product namespace references in alphabetical order.

Example:

namespace BuiltSteady.Zaplify.Website.Resources

{

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Runtime.Serialization.Json;

using System.Runtime.Serialization;

using System.Web.Security;

using Newtonsoft.Json;

using BuiltSteady.Zaplify.Website.Models;

using BuiltSteady.Zaplify.ServerEntities;

using BuiltSteady.Zaplify.ServiceHelpers;

…

}

## JavaScript

## T-Sql

# Debugging

## WinPhone Client

To debug both the web server and winphone client at the same time, you should not use F5 to launch the startup project. Instead, use right-click, debug, start new instance on the website project first. Once the browser is launched, use right-click, debug, start new instance on the winphone project. The phone emulator will launch and the winphone application will start. One can exit the phone application, but the emulator SHOULD NOT be shutdown. To restart the winphone application for debugging, simply right-click, debug, start new instance on the winphone project again.

# Deployment

## Domain Management

Domain registration for both builtsteady.com and zaplify.com is managed at <http://www.whois.com>. Login using the corporate email address, [builtsteadysoftware@gmail.com](mailto:builtsteadysoftware@gmail.com) and click on the “My Account” link.

In the “Manage Free Services” section, enter the domain name you want to manage and select the service you want to manage ( DNS mapping, email forwarding, etc. )

## Azure Accounts

BuiltSteady Inc. is registered as a Microsoft BizSpark member, providing access to free software and an Azure account with limited resources available for up to eight developers. There are currently four developer accounts registered with BizSpark ( <http://www.microsoft.com/bizspark> ) and Windows Azure ( <http://windows.azure.com> ).

[stevemillet@comcast.net](mailto:stevemillet@comcast.net), [ogazitt@hotmail.com](mailto:ogazitt@hotmail.com), [dev1@buildsteady.com](mailto:dev1@buildsteady.com), [dev2@builtsteady.com](mailto:dev2@builtsteady.com)

Login into Windows Azure using one of these registered Live IDs to access the hosted services, storage accounts, and SQL Azure used to deploy the product.

## Deployment Conventions

There will always be two deployments that are running, a production and a staging deployment. Each of these deployments will also have a dedicated branch (see Source Control Branching Conventions). In addition, there may be one or more development deployments being used by individual developers. As stated in the previous section, there are currently four subscriptions registered with BizSpark that may be used for a deployment. These Azure Accounts are currently allocated for the following deployment purposes:

Production Deployment : BizSpark – Omri [ogazitt@hotmail.com](mailto:ogazitt@hotmail.com)

Staging Deployment : BizSpark – Steve [stevemillet@comcast.net](mailto:stevemillet@comcast.net)

Development Deployment : BizSpark – Dev1 [dev1@buildsteady.com](mailto:dev1@buildsteady.com)

Development Deployment : BizSpark – Dev2 [dev2@buildsteady.com](mailto:dev2@buildsteady.com)

## SQL Azure

### DAC

The database projects produce DAC packages which can be used to deploy to SQL Azure. Once initially deployed, a database may be upgraded, retaining the existing data in the database. Use SSMS to deploy a DAC package to SQL Azure. Connect SSMS to the SQL Azure server and select Data-tier Applications under Management and right-click to deploy a new Data-tier Application. Once deployed, one may right-click an existing Data-tier Application to either delete or upgrade the deployment.

NOTE: Currently there are issues trying to upgrade a SQL Azure database using a DAC package. An alternative is to generate a script to extract the data from the current database and apply that to a new database. This assumes the schema for the current and new database are compatible.

### Backup

Before upgrading any SQL Azure database, one should create a backup copy which can be restored in case there are problems during the upgrade process. Using SSMS or the SQL Azure management console, attach to the master database for the server and run the following SQL command:

CREATE DATABASE {backup} as copy of {target}

The process of copying the database may take a while depending on the size of the database being backed up. To check the status of the backup database, attach to the master database for the server and run the following SQL command:

SELECT DATABASE name, state\_desc from sys.databases

The backup database state will change from COPYING to ONLINE when the copy is complete.

### Generating Data Script

One can generate a script which will extract the data from one database to apply to another that has a compatible schema. In SSMS, select the database to extract the data from and right-click, Generate Scripts… This will launch the wizard for generating a database script.

* Choose to script the entire database and hit the Next button
* Enter the name of the file to save the script into and hit the Advanced button
* Set Convert UDDTs to Base Types to ‘True’
* Set Script for database engine type to ‘Sql Azure Database’
* Set Types of data to script to ‘Data only’
* Hit the OK button and the Next buttons to generate the script

The generated script can now be applied to a new database to populate it with the data extracted from the original database, assuming the schema of the new database is compatible.

### Rename

When upgrading, it may be useful to rename an existing SQL Azure database. To rename a database, attach to the master database for the server and run the following SQL command:

ALTER DATABASE {current\_name} MODIFY NAME = {new\_name}

## dotnet-Host

dotnet-Host is the shared hosting service we use for the builtsteady.com website. The account is registered under [ogazitt@builtsteady.com](mailto:ogazitt@builtsteady.com) and payment is linked to a paypal account by the same name. The fee is $2.00/month for a monthly hosting program which includes ASP.NET and some basic SQL (we only use the ASP.NET hosting).

DNS management for the builtsteady.com domain is done through whois.com. The A record points to the IP address for the shared hosting service.

The dotnet-Host.com [control panel](https://panel.dotnet-host.com/Default.aspx?pid=Home) login is builtste.

The website is actually hosted at <http://builtsteady.com.dotnet-host.com/>

The MSDeploy service runs at deploy0920.dotnet-host.com:8172 with the site being builtsteady.com and the deployment Username being builtsteadycom\_dep (system-generated).

To find the publishing profile for MSDeploying an ASP.NET project into this hoster, login to the control panel, go to Websites, click builtsteady.com, click the Web Publishing tab. There is a link for downloading the publishing profile, which can be added/imported into Visual Studio.

## iOS Device

Steps for deploying iOS application to an iOS device.

* Plug-in device to a Mac iOS computer with Xcode installed. (unlock device if necessary)
* Run Xcode and go to Organizer window
* Under devices, the device should be displayed
* Click, “Use for Development” button
* Sign-in with Apple Developer ID ( builtsteadysoftware@gmail.com )
* Submit request for distribution certificate via Xcode
  + This should have created a provisioning profile for your device
* Go into MonoDevelop and open the iOS solution
* Set the active configuration to “Release | iPhone”
* Hit run (which will build and deploy to device)
  + You may need to configure build for specific iOS version (4.3 or 5.0)

# Test