# Version History

|  |  |  |
| --- | --- | --- |
| Version | Date | Changes |
| 0.0.1 | 20 March 2012 | Initial version following creation of a “walking skeleton” |
| 0.0.2 | 13 August 2012 | Updated after Jenkins CI server was ready. |

# Table of Contents

Contents

[1. Version History 1](#_Toc332618525)

[2. Table of Contents 2](#_Toc332618526)

[3. Overview 4](#_Toc332618527)

[3.1. System Map - Workflow 4](#_Toc332618528)

[3.2. Wireframes and Interface descriptions 4](#_Toc332618529)

[4. Development Environment 5](#_Toc332618530)

[5. Developers FAQ 6](#_Toc332618531)

[5.1. How do I get started? 6](#_Toc332618532)

[5.2. How do I apply a change to the BSO database? 7](#_Toc332618533)

[5.3. How do I deploy my changes? 8](#_Toc332618534)

[5.4. Where are the error logs? 8](#_Toc332618535)

[5.4.1. Log4Net 8](#_Toc332618536)

[5.4.2. Elmah 9](#_Toc332618537)

[5.5. How do I debug a problem found on UAT? 9](#_Toc332618538)

[5.6. What bits of code exemplify “best practice”? 9](#_Toc332618539)

[6. Testing 11](#_Toc332618540)

[6.1. Overview 11](#_Toc332618541)

[6.2. Test Categories 11](#_Toc332618542)

[6.2.1. Smoke Tests 11](#_Toc332618543)

[6.2.2. Acceptance Tests 11](#_Toc332618544)

[6.2.3. Service Integration Tests 12](#_Toc332618545)

[6.2.4. Database Integration Tests 12](#_Toc332618546)

[6.2.5. JavaScript Unit Tests 12](#_Toc332618547)

[6.2.6. Unit Tests 13](#_Toc332618548)

[6.2.7. Test Automation 13](#_Toc332618549)

[7. Architecture 15](#_Toc332618550)

[7.1. System Architecture 15](#_Toc332618551)

[7.1.1. Application Configuration 15](#_Toc332618552)

[7.2. Application Architecture 17](#_Toc332618553)

[7.2.1. Production Technology Stack 17](#_Toc332618554)

[7.2.2. Supporting Technology 17](#_Toc332618555)

[7.2.3. Deployed Configurations 18](#_Toc332618556)

[7.2.4. Other Technologies Considered 22](#_Toc332618557)

[7.3. Hardware Requirements 22](#_Toc332618558)

[7.4. Software Requirements 22](#_Toc332618559)

[8. System Design 23](#_Toc332618560)

[8.1. Business Services 23](#_Toc332618561)

[8.2. Data 23](#_Toc332618562)

[8.2.1. Key Database Relationships 23](#_Toc332618563)

[8.2.2. DBDeploy 27](#_Toc332618564)

[8.2.3. Automatically Generated Practice Code 27](#_Toc332618565)

[8.2.4. NHibernate 28](#_Toc332618566)

[8.2.5. Managing Configurable Data in the TMS Database 28](#_Toc332618567)

[8.2.6. Transferring Direct Debit Payments 30](#_Toc332618568)

[8.3. Security 31](#_Toc332618569)

[9. Interfaces 32](#_Toc332618570)

[9.1. Interfaces to Other Systems 32](#_Toc332618571)

[9.1.1. Document Library 32](#_Toc332618572)

[9.1.2. Document Manager 32](#_Toc332618573)

[9.2. External Interfaces 33](#_Toc332618574)

[9.2.1. User Interfaces 33](#_Toc332618575)

[9.2.2. Hardware Interfaces 33](#_Toc332618576)

[9.2.3. Software Interfaces 33](#_Toc332618577)

[9.2.4. Communication Interfaces 34](#_Toc332618578)

[10. Security 35](#_Toc332618579)

[11. Additional Specification 36](#_Toc332618580)

[11.1. Performance 36](#_Toc332618581)

[11.2. Business Continuity 36](#_Toc332618582)

# Overview

This document is updated as the BusinessSafe development progresses. As such, at any given moment it will reflect development to date and may include some design/specification for imminent developments but will not have the “full picture” until BusinessSafe development is complete.

Sections that relate to development that has not yet been completed are shown in blue.

## System Map - Workflow

<<Text>>

## Wireframes and Interface descriptions

<<Text>>

# Development Environment

Prerequisites for developing BSO:

* PC running MS Windows
* Visual Studio 2010 Professional Edition
* SQL Server 2008
* Mercurial (VCS) with TortoiseHg and Kiln Mercurial extensions
* NServiceBus

*See also “How do I get started?” (5.1)*

# Developers FAQ

## How do I get started?

* Install ruby from below link:

<http://rubyinstaller.org/downloads/>

* Get PenisulaRuby project from Kiln

<http://fogbugz/kiln/Code/Repositories/Internal-Applications/PeninsulaRuby>

* Configure Internet Explorer to enable active content to run in files on the local computer:   
  From IE, select *Internet Options*, click on the *Advanced* tab and on the *Settings* pane, select the Security node. Ensure that *Allow active content to run in files on My Computer* is checked.
* Create a folder to hold the BusinessSafe source code. Because of issues regarding the way that Nant interacts with the Windows command prompt, it’s important that the **folder name doesn’t contain spaces**.
* Default folder in Rake script is: D:\Inetpub\BusinessSafe
* Get a local clone of the Mercurial BusinessSafe repo from Kiln at

<http://pbsdevtools1.hq.peninsula-uk.local:8054/kiln/Repo/Repositories/External-Sites/BusinessSafe>

* Create a folder to hold the PeninsulaOnline source code. Because of issues regarding the way that Nant interacts with the Windows command prompt, it’s important that the **folder name doesn’t contain spaces**.
* Default folder in Rake script is: D:\Inetpub\ PeninsulaOnline
* Get a local clone of the Mercurial PeninsulaOnline repo from Kiln at

<http://pbsdevtools1.hq.peninsula-uk.local:8054/kiln/Repo/Repositories/Group/PeninsulaOnline>

* Create a folder to hold the ClientDetailsServices source code. Because of issues regarding the way that Nant interacts with the Windows command prompt, it’s important that the **folder name doesn’t contain spaces**.
* Default folder in Rake script is: D:\Inetpub\ ClientDetailsServices
* Get a local clone of the Mercurial ClientDetailsServices repo from Kiln at

<http://pbsdevtools1.hq.peninsula-uk.local:8054/kiln/Repo/Repositories/Internal-Applications/ClientDetailsServices>

* Make sure you have an instance of SQL 2008 R” installed. The instance name should be:

(local)\sql2008r2

* Run rake dbsetup command to add three projects
* *All connection strings are stored in Machine.Config and can be found on:*

C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config

These are needed connection strings:

<connectionStrings>

<add name="BusinessSafe" connectionString="Data Source='(local)\sql2008r2';Initial Catalog=BusinessSafe;Min Pool Size=2;Max Pool Size=60;Connect Timeout=60;User Id=intranetadmin;Password=intadpas;" providerName="System.Data.SqlClient" />

<add name="Peninsula\_BusinessSafe" connectionString="Data Source='(local)\sql2008r2';Initial Catalog=Peninsula\_BusinessSafe;Min Pool Size=2;Max Pool Size=60;Connect Timeout=60;User Id=intranetadmin;Password=intadpas;" providerName="System.Data.SqlClient" />

<add name="PeninsulaOnline" connectionString="Data Source='(local)\sql2008r2';Initial Catalog=PeninsulaOnline;Min Pool Size=2;Max Pool Size=60;Connect Timeout=60;User Id=intranetadmin;Password=intadpas;" providerName="System.Data.SqlClient" />

<add name="HRO" connectionString="Data Source='PBSUATSQL2\Uat2';Initial Catalog=HRO;Min Pool Size=2;Max Pool Size=60;Connect Timeout=60;User Id=intranetadmin;Password=intadpas;" providerName="System.Data.SqlClient" />

<add name="PeninsulaEntities" connectionString="metadata=res://\*/Peninsula.csdl|res://\*/Peninsula.ssdl|res://\*/Peninsula.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=(local)\sql2008r2;initial catalog=Peninsula\_BusinessSafe;integrated security=True;multipleactiveresultsets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

</connectionStrings>

* Open <MyClone>/BusinessSafe4/TaxWise.sln in VS2010
* Build – *should build the code plus create a local test installation of the TMS databases if you’ve not already got one*
* Run all tests with category Unit – *should all pass*
* Run all tests with category Acceptance – *should all pass*

## How do I apply a change to the BSO database?

* You need to write a DBDeploy script. Read section 8.2.2. If the change involves a change to standing data, read section 8.2.5.
* Look at the examples in <MyClone>/SQLScripts/BusinessSafe. Please note that later scripts are likelier to demonstrate best practice than earlier ones.
* Create your DBDeploy script and test it (including the “undo” section) on your local, test BusinessSafe database.
* Run Rake DBSetup command to run all the DBDeploy scripts including your new one.
* Test.
* Check-in.

## How do I deploy my changes?

* Do your changes locally; test them; commit them to your local repository.
* Push your changes to the development Kiln repository.
* Wait for the CI to build and test your changes.
* With agreement from the Product Owner, push your change to UAT using the CI. Test.
* Make sure that the correct version of the Access Database Engine is installed on the destination machine. The live servers are currently running a 64 bit version of Windows Server 2008 R2 so you will need to deploy the database engine using the 64 bit installer (AccessDatabaseEngine\_x64.exe). Both 32 and 64 bit versions of the installers can be found here:

[\\pbsw23it\it\Development](file:///\\pbsw23it\it\Development) Resources\Microsoft Access Database Engine 2010 Redistributable

* With agreement from the Product Owner, push your change to Live using the CI. Test.

## Where are the error logs?

### Log4Net

Logging is done using log4net to a log file TaxWise.IIS.log which can be found in the app\_data subfolder of the deployed application.

Remember - Live TMS is distributed across 3 web servers – so you’ll have to look at all 3 deployment folders:

* [\\pbsweb01\c$\inetpub\TaxWiseManagementSystem\app\_data\TaxWise.IIS.log](file:///\\pbsweb01\c$\inetpub\TaxWiseManagementSystem\app_data\TaxWise.IIS.log)
* [\\pbsweb02\c$\inetpub\TaxWiseManagementSystem\app\_data\TaxWise.IIS.log](file:///\\pbsweb02\c$\inetpub\TaxWiseManagementSystem\app_data\TaxWise.IIS.log)
* [\\pbsweb03\c$\inetpub\TaxWiseManagementSystem\app\_data\TaxWise.IIS.log](file:///\\pbsweb03\c$\inetpub\TaxWiseManagementSystem\app_data\TaxWise.IIS.log)

### Elmah

If an exception is uncaught to the point of causing the ‘oops’ page or the yellow screen of death, it will get logged to Elmah. To see the exception log, simply add Elmah.axd to the end of the taxwise URL. For instance, the URL from the UAT environment would be <http://pbswebtest2:8062/elmah.axd>

## How do I debug a problem found on UAT?

* Make sure you have a TMS build that’s compatible with the UAT databases (be careful for example if recent DB changes have not yet made it to UAT).
* Set the config files so that your build references the UAT databases by running ConfigureAsUAT.bat
* Run TMS under the debugger and investigate the issue
* Remember to reset your config back (ConfigureAsDevTest.bat) or rollback the config changes before checking in!

## What bits of code exemplify “best practice”?

Some older TMS code really ought to be refactored – they are perhaps not the best example of how to develop TMS code. So if you want to use some code as a “template” for your change, you might want to look at two or three examples first. Here are some “best practice” examples from various areas of the code base:

* TaxWiseManagementSystem
* TaxWiseManagementSystemTests
* TaxWise.Services: Classes in ClientList folder
* TaxWise.Services.Tests: Tests in ClientList folder
* TaxWise.Infrastructure: (any)
* TaxWise.Infrastructure.Tests: (any)
* TaxWise.Domain: Classes in ClientList folder
* TaxWise.Domain.Tests: Tests in ClientList folder
* TaxWise.Data: Classes in ClientList folder
* TaxWise.Data.Tests: Tests in ClientList folder
* AcceptanceTests

# Testing

## Overview

TaxWise is developed using TDD. It comes with an extensive suite of tests at several levels:

* Smoke Tests
* Acceptance Tests
* Unit Tests
* Integration Tests

To maintain the quality of the code, tests must be added at all appropriate levels whenever changes are made. Because there is a lot of test code, it is very important that the test code is maintained to a high quality and time is taken to refactor it as necessary.

We use a number of supporting test technologies:

* + We use mstest as our default unit test framework.
  + WatiN is a framework that allows a test to launch a browser (IE) and drive the system using its User Interface.
  + Jasmine is a framework for unit testing JavaScript code.

## Test Categories

### Smoke Tests

A small number of end-to-end acceptance tests (see next section) have been designated Smoke tests. These run in a reasonably short time (less than 2 minutes) and provide an indication that the TMS system is functioning correctly. The intention is that the smoke tests should include at least 1 example of every area of TMS functionality (“broad but shallow tests”).

Developers may freely commit to their local Mercurial repository but all smoke tests must be run before any change is committed to Kiln. [7.2.2]

Smoke Tests do not run under CI.

Implementation:

* Smoke tests use the WatiN test framework to drive the browser to exercise the TMS system. These tests are launched from the mstest unit test framework with tests annotated TestCategory("Smoke")
* Smoke tests can be run from within Visual Studio by selecting those with category Smoke, or by running MinimumRequiredCheckinTests.bat
* See Acceptance Tests for more implementation details.

### Acceptance Tests

Full end-to-end tests are available for nearly all the functionality. *These tests take a long time to run (up to half an hour) and can sometimes be brittle. If you find that an acceptance test fails, try re-running it!*

Acceptance Tests should be run before any major change is submitted to the CI server. When checking in a minor change, select those tests that exercise the modified functionality.

Acceptance Tests do not run under CI.

Implementation:

* Acceptance tests use the mstest unit test framework with tests annotated TestCategory("Acceptance")
* Acceptance tests use the WatiN framework to drive a browser to exercise the TMS system
* We use [the Page class model](http://watinandmore.blogspot.com/2009/06/introducing-page-class.html) to structure the representations of the page controls (see folder AcceptanceTests/PageObjectModels)
* We use View Models to present an abstraction of the UI “objects” (see folder AcceptanceTests/ViewModels)
* The TMS system must have been deployed to a local IIS server before the tests are run (so run Deploy.bat first).
* A large number of low-level helper extension methods have been written for interacting with the controls. Please use and extend these (see folder AcceptanceTests/Helpers)
* Best practice for writing and structuring acceptance tests can be found by looking at the Smoke Test.
* A local, test TMS database is used – this is rebuilt when Deploy.bat is run. If necessary, the database may be rebuilt without redeploying by executing the RecreateDatabase.bat script.
* Acceptance tests can be run from within Visual Studio by selecting those with category Acceptance, or by running AcceptanceTests.bat

### Service Integration Tests

Service Integration tests exercise all production code from the service layer down. This means that it **excludes** the web site UI code (TaxWiseManagementSystem project).

These tests must be run before any change is checked in. These tests are run by the CI server.

Implementation:

* Service Integration tests use the mstest unit test framework with tests annotated TestCategory("Integration")
* These tests rely on the existence of a test instance of the TaxWiseManagementSystem database populated with a specific set of data. This is rebuilt when Deploy.bat is run.
* Integration tests can be run from within Visual Studio by selecting those with category Integration, or by running MinimumRequiredCheckinTests.bat.

### Database Integration Tests

Database Integration tests exercise code that integrates with NHibernate for DB persistence.

All integration tests must be run before any change is checked in. These tests are run by the CI server.

Implementation:

* Integration tests use the mstest unit test framework with tests annotated TestCategory("Integration")
* These tests rely on the existence of a test instance of the TaxWiseManagementSystem database populated with a specific set of data. This is rebuilt when Deploy.bat is run.

Integration tests can be run from within Visual Studio by selecting those with category Integration, or by running MinimumRequiredCheckinTests.bat.

### JavaScript Unit Tests

JavaScript unit tests use the Jasmine BDD framework to test client-side (browser) code.

These tests must be run before any change is committed to Kiln. These tests are run by the CI server.

Implementation:

* Integration tests use the mstest unit test framework with tests annotated TestCategory("Integration") to run Jasmine tests.
* Integration tests can be run from within Visual Studio by selecting those with category Integration, or by running MinimumRequiredCheckinTests.bat.
* These tests must be run as an administrator (so make sure you launch Visual Studio with admin rights)

Hints and Tips:

* You can open the SpecRunner.html file from any browser and the tests will immediately be run as part of the page load.
* Do not attempt to diagnose failing tests by running the unit test within Visual Studio – open the SpecRunner.html from within a browser and debug/diagnose as appropriate.

### Unit Tests

All code except for some UI level (ASP.NET) code is subject to unit tests. Testing framework is mstest.

In this context, we conform to the strict definition of unit tests: a single unit (class) with all dependencies mocked.

All unit tests must be run before any change is checked in. These tests are run by the CI server.

Implementation:

* Unit tests use the mstest unit test framework with tests annotated TestCategory("Unit")
* Unit tests can be run from within Visual Studio by selecting those with category Unit, or by running MinimumRequiredCheckinTests.bat.
* Most tests use the Moq framework as the developers found it more usable in edge cases than Rhino Mocks (some older tests still use Rhino Mocks).

### Test Automation

A batch script has been provided that can be used in scheduling our acceptance tests to run overnight. It’s called ScheduledTestRunner.bat and it’s in the top level TaxWise folder in Kiln.

These are the steps that you need to perform on your own workstation in order to use it:

1. Pull ScheduledTestRunner.bat from Kiln
2. Set up a scheduled task. Give it a name and use the defaults but do add the following under the *Actions* tab:  
   Action: Start a program  
   Program/Script: cmd.exe  
   Add Arguments: /c ScheduledTestRunner.bat MinimumRequiredCheckinTests **1**  
   Start In: F:\Code\TaxWiseManagementSystem **2**
3. On the *General* tab for your new task, tick the *Run with highest privileges* checkbox
4. Test the job. The tests should run and at the end, notepad should open to show the test results **3**
5. If everything worked, edit the scheduled task and replace MinimumRequiredCheckinTests with AcceptanceTests
6. Set up an appropriate daily overnight trigger on the *Triggers* tab

Notes

1. Were just using MinimumRequiredCheckinTests whilst verifying that the setup works
2. F:\Code\TaxWiseManagementSystem is just an example. If your TaxWise codebase is in a different place, you’ll need to modify the *Add Arguments* property accordingly
3. The scheduled task will not finish until after you close Notepad

# Architecture

## System Architecture

Business runs as an ASP.NET MVC application hosted on IIS. It uses data stored in 3 separate SQLServer databases. The web application is only available on the Peninsula intranet. The current live configuration is detailed in section 0.

**Application Configuration**

Business data is stored in the SQLServer databases. The application can be configured by the web.config file included in the installation. Note that this file is reinstalled as part of the automated deployment and so any configuration changes must be automated and checked into the TMS source code repository.

#### Log4Net

TMS uses Log4Net for diagnostic logging. See <http://logging.apache.org/log4net/> for full details on log4Net configuration.

Do not log at DEBUG level in “live” as it produces lots of log messages and slows the system down. The TaxWise.build Nant script will automatically set the appropriate log4net configuration in web.config when it deploys to dev, CI, UAT or live.

#### AppSettings

The following application settings are included:

|  |  |
| --- | --- |
| **Key** | **Notes** |
| Telerik.Skin | ***Unknown*** |
| LegacyTaxWiseSupportEnabled | Enables or disables support for duplicating practice, policy and user changes in the legacy TaxWise database. |
| LegacyTaxWiseDefaultExcess | Excess assigned to a newly created practice in the legacy TaxWise database. |
| LegacyTaxWiseDefaultTaxWisePlus | Setting for Taxwise Plus assigned to a newly created practice in the legacy TaxWise database |
| LegacyTaxWisePolicyStatusIdCorrespondingTo  PolicyStatusConcluded  LegacyTaxWisePolicyStatusIdCorrespondingTo  PolicyStatusLive  LegacyTaxWisePolicyStatusIdCorrespondingTo  PolicyStatusLiveNoContract  LegacyTaxWisePolicyStatusIdCorrespondingTo  PolicyStatusPending  LegacyTaxWisePolicyStatusIdCorrespondingTo  PolicyStatusQuote  LegacyTaxWisePolicyStatusIdCorrespondingTo  PolicyStatusQuoteExpired  LegacyTaxWisePolicyStatusIdCorrespondingTo  PolicyStatusVoid | Provide the mapping from the policy status to the policy status field in the legacy TaxWise database. |
| PeninsulaMainSiteAddressTypeId | Identifies the “main” site address type |
| PeninsulaDefaultPBSIndustryTypeId | PBS industry type assigned to a new practice. |
| PeninsulaDefaultLegalLocationTypeId | Legal location assigned to a new practice. |
| PeninsulaDefaultPaymentMethodId | Payment method assigned to a new practice. |
| PeninsulaDefaultCustomerGroupId | Customer group assigned to a new practice. |
| PeninsulaDefaultContactNameSoundEx | SoundEx value assigned whena new contact is created |
| PeninsulaDefaultOperaSalesLedger | Opera Sales Ledger value assigned to a new practice |
| DefaultPasswordForANewUser | Password assigned when a new user is created (as a result of creating a new contact) |
| DefaultRoleIdForANewUser | Role assigned when a new user is created (as a result of creating a new contact) |
| DefaultNewUserLegacySecurityTypeId | The security type assigned when a new user is created (as a result of creating a new contact) in the legacy TaxWise database |
| TaxWiseInternalPracticeId | The ID of the TaxWise Internal practice (to which all internal users are assigned) |
| NHibernateProfilerEnabled | Set this to “true” to diagnose DB performance issues |
| DocumentLibraryDownloadHoldingPath | Temporary location of files as they are downloaded from the Document Library |
| DocumentLibraryUploadHoldingPath | Temporary location of files as they are uploaded to the Document Library |
| DocumentLibraryTaxWiseApplicationId | Id of the TaxWise application in Document Library |
| DocumentManagerHoldingPath | Temporary location of files as they are generated by the Document Manager |
| HoldingPathCredentialsDomain | When documents are written to one of the holding path folders, TMS impersonates this account. The password is encrypted using a RijndaelEncryptor. |
| HoldingPathCredentialsUsername |
| HoldingPathCredentialsEncryptedPassword |
| Globalization | Sets the uiCulture to English, and the culture to English-British. This enforces how we display dates, currency values etc. |

## Application Architecture

### Production Technology Stack

* ASP.NET Web Application hosted on IIS
* Browser (aka client) code uses Telerik controls, JavaScript and the jQuery JavaScript library
* Application is structured as an n-tier app following principles outlined in <http://lucidcoding.blogspot.com/>
* Persistence layer is managed using NHibernate
* Inversion of control container via NInject
* Diagnostic logging using log4net
* SQLServer 2005 database
* Document Manager [9.1.2] and Document library [9.1.1] are accessed as web services

### Supporting Technology

* Acceptance tests invoked by MSTest using the WatiN framework
* Smoke tests invoked by MSTest using the WatiN framework
* Integration tests invoked by MSTest
* Server unit tests invoked by MSTest and using the Moq mocking framework
* Client unit tests invoked by MSTest and WatiN to drive tests written using Jasmine JavaScript unit test library plus jasmine-jQuery extensions.
* Version control system: Mercurial at <http://pbsdevtools1.hq.peninsula-uk.local:8054/kiln/Repo/Repositories/Group/TaxWiseManagementSystem>
  + - Development was originally in Visual SourceSafe (vss on pbsw23drvm at $/Source files/TaxWise).
    - Legacy TaxWise Online source code is at $/Source files/TaxWiseOnline)
* Database creation/upgrade scripts managed using DBDeploy (see section 8.2.2)
* Build and test scripts: Nant (TaxWise.build)
* Continuous Integration: Cruise Control.NET (<http://pbscihro1/ccnet/server/local/ViewServerReport.aspx?Category=TaxWise>)

### Deployed Configurations

#### Development Config

Developer will usually use Visual Studio 2010 to build and test, but can use scripts to build, test or deploy. The following convenience rake file can be used to build, test and run the acceptance test on the dev machine:

* Rake Build – builds the project
* Rake Test – Runs all the unit tests
* Rake AcceptanceTest – Runs all the Acceptance tests
* Rake Push – Gets the latest version of the code from CI, Builds it and then runs unit tests and acceptance tests
* Rake DBSetup – Will rebuild the database from scratch

The following shows the configuration used by the developers on desktop development machines:

IIS: localhost

Peninsula Online Maintenance

Peninsula Online

ClientDetailsService

BusinessSafe

DB Server: (local)\sql2008

Peninsula Online

Peninsula \_BusinessSafe

BusinessSafe

#### BSO Jenkins CI Configuration

Jenkins CI server rebuilds a “dev” build on every check in:

* All unit tests and integration tests are run.
* *Acceptance tests are run.*
* All databases are rebuilt.

You can access Jenkins web dashboard by going to following URL: <http://pbsuatbsoci1:8080/view/CI/>

We have four main web applications on the CI Server.

* **ClientDetailsServices** is a collection of REST services to access Client and Agreement Data on Peninsula database.

<http://pbsuatbsoci1:8072/restservice/v1.0/client/18268>

* **PeninsulaOnline.Web** contains the marketing website and login page for our online tools.

<http://pbsuatbsoci1:8076/>

* **Peninsula.Maintenance** is the internal tool to manage users and clients.

<http://pbsuatbsoci1:8074/>

* **BusinessSafe** is the external facing BusinessSafe application.

<http://pbsuatbsoci1:8070/>

<http://pbsuatbsoci1:8080/>

The following configuration is set up and used:

IIS: pbsuatbsoci1, website TaxWise

Peninsula Online Maintenance

Peninsula Online

ClientDetailsService

BusinessSafe

DB Server: PBSUATSQL01

Peninsula Online

Peninsula \_BusinessSafe

BusinessSafe

The support team have created a template Jenkins server with all the necessary tools to start using Jenkins.

All configuration for Jenkins jobs are accessible in \\pbsuatbsoci1\c$\Program Files (x86)\Jenkins\jobs folder.

Jenkins poles the code repository every minute and builds a new version of each project if a newer version is available. At the moment we build the code on the server using Jenkins and rakeFile\_PBSUATBSOCI1 rake script will run the tests and deploy the code to the serve.

**CI Links:**

You can find the links to CI server on following page:

<http://pbsuatbsoci1:8080/job/BusinessSafe/ws/BSOLinks/BSOCI.html>

This page is lives in BusinessSafe\BSOLinks folder. If any changes are needed BSOCI.html or BSOUAT.html can be edited and pushed.

#### BSO Jenkins UAT Configuration

Jenkins CI server rebuilds a “UAT” build on demand for each project (requires a manual Force of the task):

* It will upgrade the database.
* Fresh deployment to PBSUATBSO1 and PBSUATBSO2 servers.

The release procedure is being handled by rakeFile\_PBSUATBSO1\_PBSUATBSO2 rake script in each project and Jenkins just runs that script.

The following configuration is used for each project:

PBSUATBSO1 and PBSUATBSO2

Peninsula Online Maintenance

Peninsula Online

ClientDetailsService

BusinessSafe

DB Server: PBSUATSQL01\UAT

Peninsula Online

Peninsula \_BusinessSafe

BusinessSafe

We only upgrade the database to the latest version using DB Deploy scripts. However, we can rebuild the database manually using rake scripts DBSetup method.

**CI Links:**

You can find the links to UAT servers on following page (At this point the UAT environment doesn’t support load balancing and as a result the links point to PBSUATBSO1):

<http://pbsuatbsoci1:8080/job/BusinessSafe/ws/BSOLinks/BSOUAT.html>

#### 

# System Design

## Business Services

<<Text>>

## Data

### DBDeploy

We are using the .NET port of an open-source utility called DBDeploy (<http://sourceforge.net/projects/dbdeploy-net/> ). To quote from <http://www.build-doctor.com/2010/01/17/dbdeploy-net/> :

DBDeploy is an implementation of the ActiveRecord Migrations pattern. DbDeploy.NET is the .NET port of DBDeploy. Both DBDeploy are projects initiated by ThoughtWorks. ActiveRecord comes to us via DHH.

**Why would I use it?**

When you’re developing software that hasn’t been released, the database is easy: you can tear it down and rebuild it at will. Once you have production data that people are using it, what do you do? How do you manage the change? The Migrations pattern allows you to make bite-sized changes to your database, and test it. It works very well with Continuous Integration.

It’s ideal for greenfield agile projects where you are using Continuous Integration and want to make sure that changes to the database schema will be applied to integration tests. You can use other approaches if you have an ORM and you haven’t released to production yet

The idea behind DBDeploy is very simple. The target database must contain a changelog table. DBDeploy uses this to decide what SQL scripts have not yet been run on that database. It then creates a script which concatenates all the modifications in those scripts that have not yet been run and updates the changelog.

Our use of DBDeploy:

* We are invoking DBDeploy via a Nant task (included in the dbdeploy.net distribution) in DBDeploy\_Local.build.
* Rake DBSetup method runs DBDeploy, to update the database
* When creating a brand new database, rake script will first tear down the old DB (if it exists) and recreate it, then run DBDeploy
* Scripts must follow a naming convention with a next-out numeric prefix
* A DBDeploy script can include an “undo” section which must follow a --//@UNDO marker in the script. We **do** use DBDeploy’s “undo” facility so scripts must include this section and the undo must be tested.

### NHibernate

We are not using Fluent NHibernate, as noted above. So mappings from C# classes to database entities are defined in NHIbernate mapping files (\*.hbm.xml).

We use structure map to manage NHibernate Session Factory and NHibernate Session. Structure Map will create one Session Factory for the application and will create a session for each http request. All the settings for structure map can be found in DataRegistry.cs class in BusinessSafe.Data project.

Below you can see some sections of structure map configuration from DataRegistry class:

ForSingletonOf<IBusinessSafeSessionFactory>().Use<BusinessSafeSessionFactory>();

For<IBusinessSafeSessionManager>().HybridHttpOrThreadLocalScoped().Use<BusinessSafeSessionManager>();

# Interfaces

## Interfaces to Other Systems

### Document Library

TMS uses Document Library web service for storage of user documents. Although these documents are stored in the Document Library’s own stores, a summary record is held in the TMS database (see table PolicyDocumentSummary). This summary record includes the Document Library identifier for a document.

At the time of writing:

* Document library code is in VSS at Source files/Peninsula.DocumentLibrary  
  Comes with some tests that should run out-the-box except may need to set up a local path
* UAT #1 (*this is the one we’re using*):
  + URL: <http://pbswebtest1:8056/DocumentLibraryService.svc>
  + DB: pbsuatsql1
  + Filestore: \\pbsuatweb1\holdingpath
* UAT #2:
  + URL: <http://pbswebtest2:8056/DocumentLibraryService.svc>
  + DB: pbsuatsql2
  + Filestore: [\\pbsuatweb1\holdingpath](file:///\\pbsuatweb1\holdingpath) (yes, same place!)
* Live:
  + URL: <http://pbsdoclibrary:8056/DocumentLibraryService.svc>
  + DB: pbsw23sql1/DocumentLibrary

### Document Manager

TMS uses the Document Manager 2.0 web service for generation of a document from an MS Word template. The generated document is then stored in Document Library.

An external Document Manager app (<http://pbsweb01:8067/wcf/default.aspx>) is used by TaxWise users to manage the templates within Document Manager. A **UAT version** of this application runs at <http://pbswebtest1:8067/wcf/default.aspx>

Document Manager’s database includes a BusinessObjects table. This is populated with XML representing the structure of a TMS Practice & Policy “business object” (a script is maintained in TMS at SQLScripts\DocumentManager\SetUpTaxWiseBusinessObjects.sql to set up this record in the Document Manager database).

When a user makes a request to generate a document for a policy, the Practice and Policy are translated into Business Object XML that is then sent to Document Manager. The resulting document is returned from the web service.

Document Manager also supports tables of data to be sent from the client (TMS). *More detail to be added post-implementation.*

Identifying which database a given deployment of Document Manager uses

The configured database can be read from the NHibernate config file. If the Document Manager application is deployed to pbswebtest1, the configuration file will be found here:

\\pbswebtest1\c$\inetpub\DocumentManager\hibernate.cfg.xml

* UAT:
  + Endpoints:
    - <http://pbswebtest1:8066/TemplateService.svc>
    - <http://pbswebtest1:8066/FolderService.svc>
  + DB: pbsuatsql3/DocumentManager
* Live:
  + Endpoints:
    - <http://documentmanagerservices:8066/TemplateService.svc>
    - <http://documentmanagerservices:8066/FolderService.svc>
  + DB: pbsprod2sql\prod2\DocumentManager
  + Holding path: \\pbsdocman1\HoldingPath

## External Interfaces

### User Interfaces

* BDM: Business Development Manager uses the TXMS to create and manage practice records and contracts
* TaxWise Staff: staff at Hinckley will use TXMS to carry out the following functions:
  + Manage the practice accounts
  + Manage and reconcile account payments
  + Accept and manage claims
  + Make claim payments
* Practice Users will use TXMS to
  + Manage their TXMS user accounts
  + Make claims
  + Manage their document library

### Hardware Interfaces

None

### Software Interfaces

* Peninsula Database
  + The Peninsula DB is only accessed for records relating to the practice record. In the Peninsula domain this is a “client” record with an ID referred to as a “Client Account Number” or CAN
* Legacy TaxWise database (aka “TaxWise Online”)
  + During the transition from the ‘old’ TaxWise system to the new system, the legacy database must be supported.

### Communication Interfaces

Possible interfaces yet to be finalised:

* Email

# Security

<<Text>>

# Additional Specification

## Performance

<<Text>>

## Business Continuity

<<Text>>