|  |
| --- |
| Project Raindrop |
| iInterchange Systems Pvt. Ltd.  Version 1.0 |

|  |  |  |
| --- | --- | --- |
| Date : Code : | | |
| **PREPARED BY** | **REVIEWED BY** | **APPROVED BY** |
|  |  |  |

*Note: This is a privileged and confidential document of iInterchange Systems Pvt Ltd. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), without prior written consent from* ***iInterchange***

Copyright 2010 by **iInterchange** all rights reserved.

(Subject to limited distribution and restricted disclosure only)

**REVISION HISTORY**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **VERSION NO** | **AUTHOR** | **RELEASE DATE** | **CHANGE DETAILS** | | **APPROVED BY** |
| **SECTION** | **DESCRIPTION** |
|  |  |  |  |  |  |

Table of Contents

[2. Introduction 4](#_Toc288060118)

[2.1. Purpose 4](#_Toc288060119)

[2.2. Scope 4](#_Toc288060120)

[2.3. Definitions, Acronyms and Abbreviations 4](#_Toc288060121)

[3. Tear Apart 5](#_Toc288060122)

[3.1. Page Architecture 5](#_Toc288060123)

[3.2. Spliting Of dll 5](#_Toc288060124)

[3.3. Namespace Resolution 5](#_Toc288060125)

[3.4. File Structure 5](#_Toc288060126)

[4. Generation of Patch 6](#_Toc288060127)

[4.1. Validation 6](#_Toc288060130)

[4.2. Documentation 6](#_Toc288060131)

[4.3. Versioning 6](#_Toc288060132)

[5. Automation of Updates 7](#_Toc288060137)

[6. Roadmap 8](#_Toc288060138)

[7. References 9](#_Toc288060139)

# Introduction

## Purpose

Purpose of this document is to explain about the architecture and the technologies involved in the development of Project Raindrop.

## Scope

Scope of this document will be Project Raindrop

## Definitions, Acronyms and Abbreviations

|  |  |
| --- | --- |
| Tech Group | Technology Group |
| BL | Business logic |
| RDI | Raindrop Installer |

# Tear Apart

## Page Architecture

All ASP.NET pages are composed of a designer file which contains all the markup for the controls and a code behind file which has the code for separate events like page load, button click etc. Now when a request for an asp.net page is received by the IIS, it transfers the request to the aspnet\_isapi.dll ISAPI extension, the aspnet\_isapi.dll extension in turn loads the HTTP Runtime and the processing of the request starts.

When the processing of the request starts, objects required for the processing of the request like HttpContext and HttpApplication etc are instantiated and the HTTP Modules are executed against the request until the request hits the ASP.NET page HTTP Handler and then the page life cycle starts.

## Spliting Of dll

Typically the web application of a project is compiled into a single assembly which brings down the number of assemblies produced after a build is taken; this makes it easy to obfuscate the codes in the code behind files of the ASP.NET pages. But this method has a major flaw which is, whenever a change is made in the code of a single page the whole project needs to be build and the complete build needs to be installed in the live environment. This is completely unnecessary and time consuming.

If we split the web application’s assembly into per page assembly, when the code in a page is changed only that assembly and the assemblies referenced by it needs to be replaced in the live environment. This drastically reduces the time it takes for us to make a change, take the build for the change and install the change in the live environment.

## Namespace Resolution

Even though each class file in Business logic is compiled into separate dll, the namespace remains same for all Business logic dll. This way when there is change made to one of the business logic files only that business logic class and its referenced assemblies needs to be build for deployment.

## File Structure

All the aspx page in the project inherits Pagebase class and has reference for Script files and Business logic

# Generation of Patch

A patch is a set of changes which needs to be affected in the live application.

When there is a new requirement or a change to be done anywhere in the application the changes are deployed to the live environment as a patch which replaces the necessary files to effect that change or requirement.

Generating patches has a lot of advantages over doing a complete build of the application for making a change in the live application.

All the files that are changed for a particular change or bug fix are compiled into a single executable, and the patch can be installed by running this executable. The GUID of the files are used to verify the identity of the files being changed.

In the event that the changes made by a patch needs to reverted, the patch can be uninstalled which would leave the application in the state where it was before installing the particular patch.

## Validation

While installing a patch it validates whether the dependent patch is already installed and any patch not required for this is not installed. If all the validations are successful, it will be installed.

While uninstalling the patch, it validates whether any other patch depends on it. If there is no dependency for this patch, it can be removed.

## Documentation

Once the patch is installed successfully, build report is generated for the specified change and an email is sent to the concerned person.

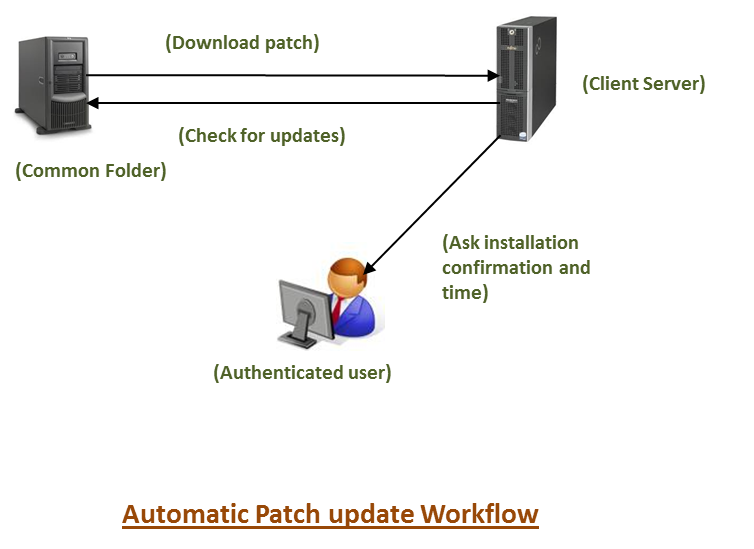
## Versioning

The project version is updated depends upon the type of update provided in patch.

# Automation of Updates

Once the patch file is created for the selective change set, it is placed in a common folder which is accessible for the customer over internet via a secured layer. Update service should be running in client server. It checks for updates periodically by monitoring the common folder. Once an update is found, the service downloads the patch and the manifest file for that patch which contains the scheduled installation time.

The user is asked to confirm the installation time. The user is allowed to change the installation time. These updates are classified into different types. It the update critical, it will be installed automatically. If the update is optional, user is asked to install the update. This reduces the down time of the application.



# Roadmap

|  |  |
| --- | --- |
| **Feature** | **Version** |
| Patch install validation | V1 |
| Patch uninstall validation | V2 |
| Documentation | V2 |
| Versioning | V2 |
| Automatic Update | V2 |

# References

* [Code Project](http://www.codeproject.com/KB/aspnet/aspnetrequestarchitecture.aspx)
* [Dot Net Slackers](http://dotnetslackers.com/articles/iis/ASPNETInternalsIISAndTheProcessModel.aspx)