Enterprise Enrollment Engine

Technical Design Document

Context and Component Dependency Injection(CCDI)

|  |  |
| --- | --- |
| project name: enterprise Enrollment engine | PlanView ID: |
| Project Manager: | |
| Project Sponsor: | |

Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Revision Number | Description of change | Author |
| 07/26/2014 | 1 | Intial Version | Shatheesh Govindarajulu, Suresh Ramasamy |
| 10/15/2014 | 2 | Updated design, usage guide and unit test case details | manjusha mukkanti, Suresh ramasamy |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1 Introduction 4](#_Toc390088038)

[1.1 Purpose 4](#_Toc390088039)

[1.2 References 4](#_Toc390088040)

[1.3 Glossary 4](#_Toc390088041)

[2 DESIGN OVERVIEW 4](#_Toc390088042)

[2.1 Module List 4](#_Toc390088043)

[2.2 Modules Interaction Flows 5](#_Toc390088044)

[2.3 Key Design Decisions 5](#_Toc390088045)

[2.4 Design Constraints and Concessions 5](#_Toc390088046)

[2.5 Design Patterns and Frameworks 5](#_Toc390088047)

[2.6 Risks and Issues 5](#_Toc390088048)

[2.6.1 Test and Code Coverage 5](#_Toc390088049)

[3 MODULE TECHNICAL DESIGN 6](#_Toc390088050)

[3.1 Context and Component Dependency Injection 6](#_Toc390088051)

[3.1.1 Design Flows 6](#_Toc390088052)

[3.1.2 Logical Component Diagrams 7](#_Toc390088058)

[3.1.3 Design Constraints 10](#_Toc390088059)

[3.1.4 Presentation Layer (Optional) 10](#_Toc390088060)

[3.1.4.1 <Screen 1..n> 10](#_Toc390088061)

[3.1.4.1.1 User Interface Layout 10](#_Toc390088062)

[3.1.4.1.2 Screen Data Mapping 10](#_Toc390088063)

[3.1.4.1.3 UI Code Implementation/Changes 10](#_Toc390088064)

[3.1.4.1.4 UI Configuration changes 10](#_Toc390088065)

[3.1.4.1.5 UI Security Changes 10](#_Toc390088066)

[3.1.4.1.6 UI Web Accessibility Changes 11](#_Toc390088067)

[3.1.5 Business Layer 11](#_Toc390088068)

[3.1.6 Data Layer 12](#_Toc390088069)

[3.1.7 Integration Layer 12](#_Toc390088070)

[3.1.8 NFR Implementation (Optional) 14](#_Toc390088071)

[3.1.9 Module Security Implementation (Optional) 14](#_Toc390088072)

[3.1.10 Environment Dependencies (Optional) 14](#_Toc390088073)

[4 BATCH (OptIONAL) 14](#_Toc390088074)

[4.1 Batch/Job Scheduling Configuration 14](#_Toc390088075)

[5 REPORTING (OPTIONAL) 14](#_Toc390088076)

[5.1 <Report 1..n> 14](#_Toc390088077)

[5.1.1 Report Layout 14](#_Toc390088078)

[5.1.2 Report Data Mapping 14](#_Toc390088079)

[5.1.3 Reports Code changes 14](#_Toc390088080)

[5.1.4 Reports Integration Code changes 14](#_Toc390088081)

[5.1.5 Report Configuration changes 14](#_Toc390088082)

[6 COMPONENT INVENTORY 14](#_Toc390088083)

[7 daTASTORE / DATABASE 14](#_Toc390088084)

[7.1 Summary of Database changes 15](#_Toc390088085)

[Refer the DDL Scripts and the schema diagram. 15](#_Toc390088086)

[7.2 Data Migration 15](#_Toc390088087)

[7.3 CCDI 15](#_Toc390088088)

[7.3.1 Schema changes 15](#_Toc390088089)

[7.3.2 Referential Integrity changes 15](#_Toc390088090)

[7.3.3 Database (DDL) Scripts Reference 15](#_Toc390088091)

[7.3.4 Data Setup 15](#_Toc390088092)

[8 Security 15](#_Toc390088093)

[9 Deployment 15](#_Toc390088094)

[9.1 Build and Packaging 15](#_Toc390088095)

[9.2 Development Model 15](#_Toc390088096)

[9.3 Dependencies 16](#_Toc390088097)

[10 ApprovalS 16](#_Toc390088098)

# 1 Introduction

## Purpose

The main goal of the this project is to develop an enterprise-wide enrollment processing platform for individual, small and large groups for WellPoint, create a low redundancy, and a flexible solution, one that can support existing system interfaces both internal / external types, and potential future system concerns, Provide short and long term strategies for moving core business from WebInsure Enrollment Manager (WEM) to Enterprise Enrollment Engine (EEE/E3) with balance and controls, Standardize inbound and outbound payload schema for each specific transaction and Develop horizontally scalable batch components to process large files of data.

The Technical Design Document (TDD) provides a comprehensive design overview of the solution at a specific point in time. It also serves as a communication medium between the **designers**, **software architects** and other **project team members** regarding design decisions which have been made on the project. The TDD is a container document for several physical design models.

The purpose of this document is to provide detailed design guideline to the development team to develop the CCDI API module.

## References



## Glossary

| **Term, Acronym or Abbreviation** | **Description** |
| --- | --- |
| EEE/E3 | Enterprise Enrollment Engine |
| WEM | WebInsure Enrollment Manager |
| CCDI | Context and Component Dependency Injection |

# DESIGN OVERVIEW

* 1. Module List

The Scope of this TDD is to discuss core aspects of CCDI components and to some extent pseudo code where it is critical to get the technical point across.

CCDI lets the container inject types defined by the application programmer, Change or extend the implementation of a bean dynamically, manage application specific properties, Environment related properties, validations that needs to be performed for a process, user messages etc.

* Context and Component Dependency Injection
  + CCDI Service Integration
  + Content Repository
  + Cache Management
  1. Modules Interaction Flows



* 1. Key Design Decisions

N/A

* 1. Design Constraints and Concessions

N/A

* 1. Design Patterns and Frameworks

CCDI framework uses Spring Integration to implement some of the enterprise integration patterns like Channel and Transformer

* 1. Risks and Issues

N/A

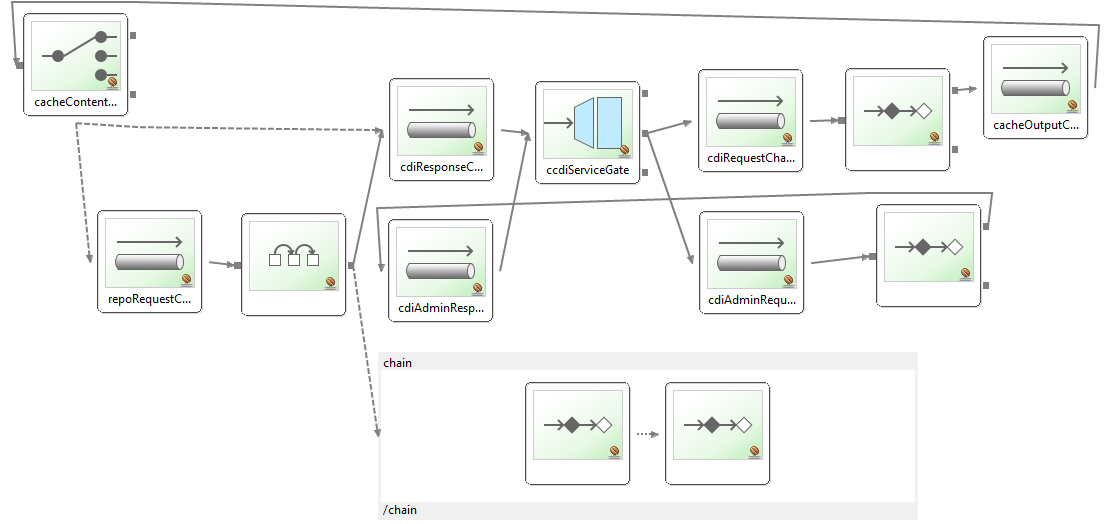
* + 1. Test and Code Coverage



# MODULE TECHNICAL DESIGN

* 1. Context and Component Dependency Injection
     1. Design Flows

CCDI Service Integration

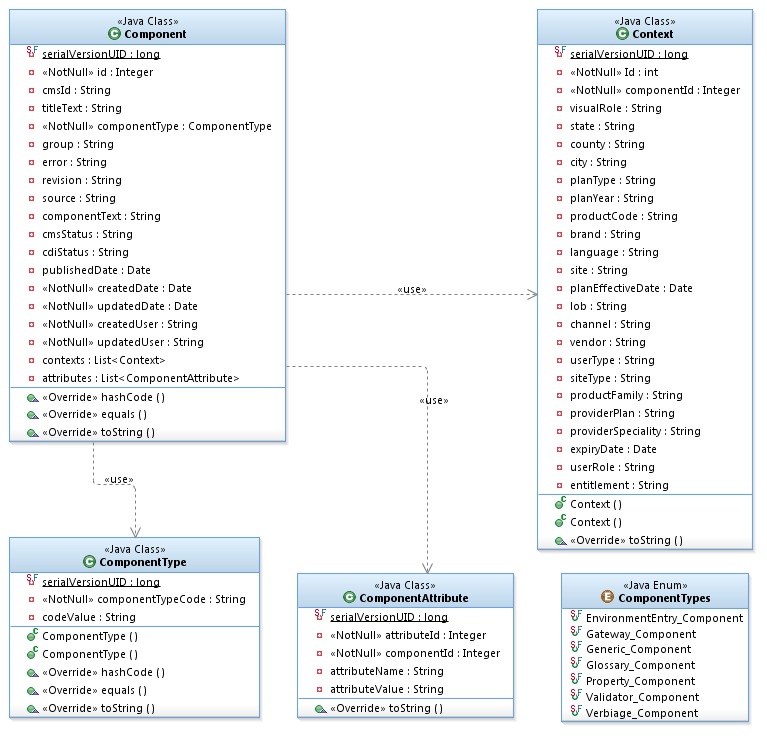
**

Sequence Diagram

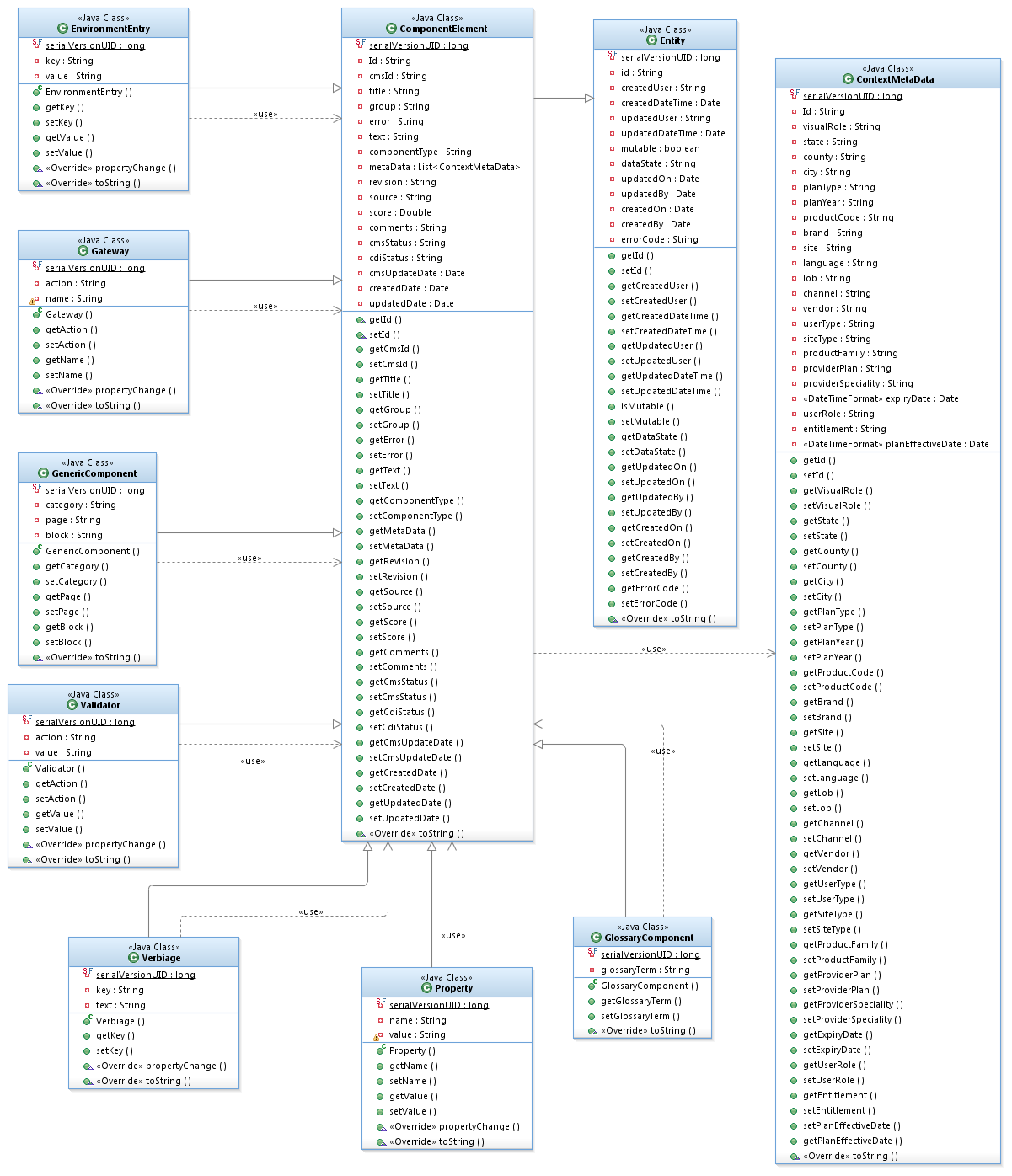


* + 1. Logical Component Diagrams

**Entities that will be used in Persistence layer**



**Entities that will be used in Business/Service layer**



**Content Repository**



* + 1. Design Constraints

N/A

* + 1. Presentation Layer (Optional)

N/A

* + - 1. <Screen 1..n>

N/A

* + - * 1. User Interface Layout

N/A

* + - * 1. Screen Data Mapping

N/A

* + - * 1. UI Code Implementation/Changes

N/A

* + - * 1. UI Configuration changes

N/A

* + - * 1. UI Security Changes

N/A

* + - * 1. UI Web Accessibility Changes

N/A

* + 1. Business Layer

The following Component types have been identified.

|  |  |  |
| --- | --- | --- |
| **Type** | **Input** | **Output** |
| Property | Name | Value |
| Environment | Key | Value |
| Gateway | Action | Name |
| Verbiage | Key | Text |
| Validator | Action | Value |
| Generic Content | Category, Page, Block | Text |
| Glossary | Glossary term | Text |

Apart from the standard input listed above, all the Component types filter the information based on the Component Metadata as well.

Based on the content requested, the request key will be formed using Key builder. This key will be used to fetch the content from the cache. If the content exists in the cache then the requested content will be returned.

If there is no content available in the cache for the key then the content will be fetched from the database and returned to the client.

If there is more than one content available in the database for a key, then the content will be sorted and only the first content will be considered. This content will be updated in the cache and the same is returned to the client.

Sorting order is performed with the following contextual information

Website address, user role code, entitlement code, user type code, visual role code, channel code, vendor code, lob code, state code, brand code, county code, city name, provider specialty code, product family code, product code, plan type code, provider plan text, plan year, plan effective date, language code, site type code and expiry date

**Key builder**

Key builder forms the key for the requested content. This key will be used to get/put the content from/in cache.

The following contextual information is used to form the key irrespective of the content type.

UserRole, Entitlement, Visual Role, State, County, City, Plan Type, Plan Year, Product Code, Brand, Site, Language, Lob, Channel, Product Family, Provider Plan, User Type, Vendor, Site Type, Provider Specialty, Expiry date, Plan effective date.

Apart from the above fields, there are specific set of fields used for each content type to form the key.

**Generic content:** Category, Page, Block

**Property content:** Name

**Environment content:** Key

**Verbiage content:** Key

**Validator content:** Action

**Gateway content:** Action

**Glossary content:** Glossary term

All the above fields are combined as a String and its hash code is prefixed with a content type specific prefix. This string will be returned as Key by Key builder.

* + 1. Data Layer

The content will be fetched from DB if the content is not available in the cache.

CcdiApiRepositoryActivator uses different repository classes to fetch the content from database. These repository classes use DAO classes and FetchContentRepository to fetch the data from database.

FetchContentRepository is an inner class defined in the DAO classes which will be used to form the query parameters, to execute the query and to map the result set to domain object.

* + ComponentApiRepository: This repository class fetches the content for Property, Gateway, Verbiage, Environment and Validator content types. Also, retrieves the list of place holder config properties from CCDI. This repository class uses ComponentSelectorDao to fetch the content.
  + CcdiApiRepository : This repository class fetches the content for Glossary content and Generic content type which is searched without any wildcard type. This repository class uses CcdiSelectorDao to fetch the content.
  + CcdiApiWildcardRepository: This repository class fetches the content for Generic content type which is searched with wildcard type. This repository class uses CcdiWildcardSelectorDao to fetch the content.
    1. Integration Layer

CCDI Service integration module defines the workflow of the CDI API application. It enables loosely coupled integration of service components. The flow diagram (see Design flows) shows the integration of service components and its messaging channels. The implementation is provided by Service Integration Framework from Spring.

* CcdiRequestChannel: Input request channel from client wrapping the CcdiRequest object.
* CacheGetActivator: Cache activator which communicates with the implementing cache service (DynaCache configured as an external service) to get the requested content from cache. The cache activator also uses a KeyBuilder to build the cache key.
* CacheOutputChannel: Output channel from cache service which wraps CcdiRequest if the requested content is not found in the cache and CcdiResponse otherwise.
* CacheContentRouter: Routes the output from Cache Service. If the type of output from cache service is a CcdiResponse, the output is routed to the output channel of CdiServiceGate gateway. If the type of output from Cache Service is CcdiRequest, then it is routed to Content Repository Service.
* RepoRequestChannel: Input channel to RepoActivator wrapping CcdiRequest
* RepoActivator: Repository Activator communicates with Content Repository Service. The content repository service is responsible for getting the content requested from the content database. The input and output to this activator is ContentElement object. It uses a transformer to transform ComponentElement to Component as Component object is used in all internal operations.
* RepoOutputChannel: Output channel from RepoActivator wrapping CcdiResponse and CcdiResponse wrapping content requested and CcdiRequest.
* cachePutActivator: CachePutService activator communicates with the CacheService to put the response content into the cache with key built by KeyBuilder using wrapped CcdiRequest object in response.
* CcdiServiceGate: Gateway to CCDI Service API. It takes CcdiRequest as input channel and CdiResponse is the output channel.

CCDI API uses DynaCache implementation provided by WAS7.0 Application server. The cache service interface allows to get/put the content for a given key.

A content key is generated from CcdiRequest using the KeyBuilder class. The KeyBuilder class uses hashing algorithm to generate key from the CcdiRequest, It Uses HashCodeBuilder provided by apache commons API.

A handle to DynaCache can be obtained by using JNDI call.

CacheService will be a singleton Spring bean. It will hold all Cache instances created in WAS Server. The jndiMapping attribute is a Map with key as application name and value as the bean id defined for jndi reference.

The method get(String applicationName, Object key) uses Application Context and loads the bean defined for appropriate DistributedMap object from jndiMapping using applicationName and gets the value from DistributedMap object using the key.

The method put(String applicationName, Object key, Object value) will load and get appropriate DistributedMap object from jndiMapping using Application Context and puts the value using the key.

**Spring Configurations**

<util:map id=*"jndiMapping"* map-class=*"java.util.HashMap"*

key-type=*"java.lang.String"* value-type=*"java.lang.String"*>

<entry key=*"GENERIC"* value=*"ccdiGenericContentCache"* />

<entry key=*"PROPERTY"* value=*"ccdiPropertyCache"* />

<entry key=*"ENVIRONMENT"* value=*"ccdiEnvCache"* />

<entry key=*"VERBIAGE"* value=*"ccdiVerbiageCache"* />

<entry key=*"GATEWAY"* value=*"ccdiGatewayCache"* />

<entry key=*"VALIDATOR"* value=*"ccdiValidatorCache"* />

<entry key=*"GLOSSARY"* value=*"ccdiGlossaryCache"* />

</util:map>

<bean id=*"ccdiGenericContentCache"* class=*"org.springframework.jndi.JndiObjectFactoryBean"*>

<property name=*"jndiName"* value=*"cache/ccdiGenericContentCache"* />

</bean>

<bean id=*"ccdiPropertyCache"* class=*"org.springframework.jndi.JndiObjectFactoryBean"*>

<property name=*"jndiName"* value=*"cache/ccdiPropertyCache"* />

</bean>

<bean id=*"ccdiEnvCache"* class=*"org.springframework.jndi.JndiObjectFactoryBean"*>

<property name=*"jndiName"* value=*"cache/ccdiEnvCache"* />

</bean>

<bean id=*"ccdiVerbiageCache"* class=*"org.springframework.jndi.JndiObjectFactoryBean"*>

<property name=*"jndiName"* value=*"cache/ccdiVerbiageCache"* />

</bean>

<bean id=*"ccdiGatewayCache"* class=*"org.springframework.jndi.JndiObjectFactoryBean"*>

<property name=*"jndiName"* value=*"cache/ccdiGatewayCache"* />

</bean>

<bean id=*"ccdiValidatorCache"* class=*"org.springframework.jndi.JndiObjectFactoryBean"*>

<property name=*"jndiName"* value=*"cache/ccdiValidatorCache"* />

</bean>

<bean id=*"ccdiGlossaryCache"* class=*"org.springframework.jndi.JndiObjectFactoryBean"*>

<property name=*"jndiName"* value=*"cache/ccdiGlossaryCache"* />

</bean>

………

………

**Usage**

In CacheService activator (or a client class), the cacheService bean will be injected.

*cacheService.get("eox\_property ", keyBuilder.generateKey(request));*

Here eox\_property is the component type and a KeyBuilder is used to generate Key out of request object.

WAS7.0 IBM console provides capability to clear/refresh cache. This can be used to manually clear CDI Content cache.

* + 1. NFR Implementation (Optional)

N/A

* + 1. Module Security Implementation (Optional)

N/A

* + 1. Environment Dependencies (Optional)

N/A

# BATCH (OptIONAL)

* 1. Batch/Job Scheduling Configuration

N/A

# REPORTING (OPTIONAL)

* 1. <Report 1..n>

* + 1. Report Layout

N/A

* + 1. Report Data Mapping

N/A

* + 1. Reports Code changes

N/A

* + 1. Reports Integration Code changes

N/A

* + 1. Report Configuration changes

N/A

# COMPONENT INVENTORY

N/A

# daTASTORE / DATABASE

* 1. Summary of Database changes

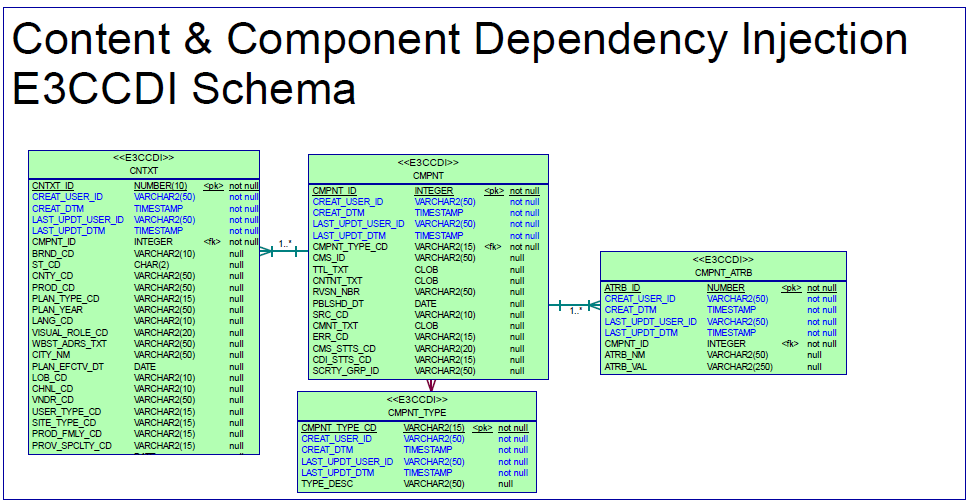
Refer the DDL Scripts and the schema diagram.

* 1. Data Migration

N/A

* 1. CCDI

## Schema changes



## Referential Integrity changes

N/A

## Database (DDL) Scripts Reference

## Data Setup

N/A

# Security

N/A

# Deployment

N/A

* 1. Build and Packaging

N/A

* 1. Development Model

N/A

* 1. Dependencies

N/A

# ApprovalS

This document should be approved by impacted stakeholders to be designated by the Project Manager and/or the Project Sponsor in compliance with the [IT Controls for Change](http://worknet.auth.wellpoint.com/resources/CDPTO4_SysDevLifeCycle/default/Intranet_Asset/PW_C117480.pdf).

| **Name** | **Title** | **Date** | **Link to email approval** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |