Department of Veterans Affairs (VA)

Benefits Claims Decision Support System (BCDSS)

Developers Guide



**May 2016**

Version 0.2

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 2016-06-29 | 0.2 | Guide Update | Dominic Yeh |
| 2016-04-20 | 0.1 | Template Created | Erik Rothwell |

Table of Contents

[1. Introduction 1](#_Toc448911457)

[1.1 Purpose 1](#_Toc448911458)

[1.2 Scope 1](#_Toc448911459)

[1.3 Intended Audience 1](#_Toc448911460)

[1.4 Assumptions and Dependencies 1](#_Toc448911461)

[2. Setting up the Local Environment 2](#_Toc448911462)

[2.1 Instructions for Building a Developer Environment 2](#_Toc448911463)

[3. Configuring and Using the FTL and JIRA Environment 2](#_Toc448911464)

[3.1 Source Code Repository Overview 3](#_Toc448911465)

[3.1.1 Tool Source Control Overview 3](#_Toc448911466)

[3.2 Checking Files into Source Code Repository 3](#_Toc448911467)

[3.2.1 Refresh Remote Changes 3](#_Toc448911468)

[3.2.2 Refresh Sandbox 3](#_Toc448911469)

[3.2.3 Check-in and Deliver 3](#_Toc448911470)

[3.2.4 Resolving File Conflicts 3](#_Toc448911471)

[3.2.5 Configuring an External Compare Tool 3](#_Toc448911472)

[3.3 Creating and Managing Work Items in JIRA 3](#_Toc448911473)

[3.3.1 Creating Work Items 3](#_Toc448911474)

[3.3.2 Adding Links, Subscribers, and Approvers 4](#_Toc448911475)

[3.3.3 Notes for Creating Work Items 4](#_Toc448911476)

[3.3.4 Updating Tasks as You Work on Them 5](#_Toc448911477)

[3.3.5 Extending Stories across Sprints 5](#_Toc448911478)

[3.3.6 Closing Out Stories 5](#_Toc448911479)

[3.4 Planning Procedures 5](#_Toc448911480)

[3.4.1 Determining Sprint Planning Tasks 6](#_Toc448911481)

[3.4.2 Determining Construction/Development Tasks 6](#_Toc448911482)

[3.4.3 Adding a Requirement 7](#_Toc448911483)

[3.4.4 Linking the Requirement to the Story 7](#_Toc448911484)

[3.4.5 Accessing Requirements from JIRA 7](#_Toc448911485)

[3.4.6 Adding a Test Case 7](#_Toc448911486)

[3.4.7 Linking Requirements to the Test Case 7](#_Toc448911487)

[3.4.8 Linking the Test Case to the Story 7](#_Toc448911488)

[3.4.9 Accessing Test Cases from JIRA 7](#_Toc448911489)

[3.5 Configuration Management Procedures 7](#_Toc448911490)

[3.5.1 Builds 7](#_Toc448911491)

[3.5.2 Automated Deployment Process 8](#_Toc448911492)

[3.5.3 CI&D Dashboard 8](#_Toc448911493)

[3.6 Moving Files in SCR 9](#_Toc448911494)

[3.6.1 Moving a File or Folder 9](#_Toc448911495)

[4. Development Guidelines 10](#_Toc448911496)

[4.1 Implementing Front-End Components Standards 10](#_Toc448911497)

[4.2 Unsaved Changes Validation 11](#_Toc448911498)

[4.3 Required Fields Validation 12](#_Toc448911499)

[4.3.1 Description 12](#_Toc448911500)

[4.3.2 Initialization 12](#_Toc448911501)

[4.3.3 Applying Field Validation 12](#_Toc448911502)

[4.3.4 Usage Tips 14](#_Toc448911503)

[4.4 Miscellaneous Notes 14](#_Toc448911504)

[4.4.1 Target Screen Resolution 14](#_Toc448911505)

[A. Terminology 15](#_Toc448911506)

[B. References 16](#_Toc448911507)

[B.1. User Documentation 16](#_Toc448911508)

[Attachment A. Approval Signatures 17](#_Toc448911509)

# Introduction

This developer’s guide provides the information required for a new member of the Department of Veterans Affairs (VA) Benefits Claims Decision Support System (BCDSS) product development team to create a local environment with the tools and configuration settings necessary to perform his or her role on the team.

Contractors develop the BCDSS application on their laptops and perform integration testing in the VACI FTL environment. The FTL team must configure a consistent image of BCDSS so that developer testing can occur in a common fashion because the BCDSS team develops the application on local contractor laptops. This Guide includes instructions for configuring a local developer, analyst, technical writer, and tester workstation.

Local environments should not have any dependencies on the VA network or the FTL Sandbox to develop, build, deploy and test BCDSS.

***Note:*** As a living document, this document will undergo continual refinement based on feedback from BCDSS team users and other evaluators.

## Purpose

This manual provides the information required for members of the BCDSS product development team to create a software environment and use the tools necessary to develop, test, manage, and document the BCDSS application. It also provides development guidelines for developers.

## Scope

This document’s scope covers the set of tasks necessary to create a local environment for developing, testing, managing, and documenting the BCDSS product.

## Intended Audience

This document’s intended audience includes personnel involved with BCDSS product development, including software development, configuration management, testing, documentation, and other functions that require setting up an environment and using the standard set of BCDSS product development tools.

## Assumptions and Dependencies

# Setting up the Local Environment

The selections below detail the instructions for configuring a local environment on a local team-member’s computer. These sections assume that local environments will not depend on the VA network or the FTL Sandbox to develop, build, deploy, and test BCDSS.

## Instructions for Building a Developer Environment

Instructions for how to download and configure all tools required to contribute to the BCDSS application can be found on the BCDSS Confluence site.

### Development Tools

Java SE Development Kit 8u91

Java Platform, Standard Edition (Java SE) lets you develop and deploy Java applications on desktops and servers, as well as in today's demanding embedded environments. Java offers the rich user interface, performance, versatility, portability, and security that today's applications require.

Ant 1.9.4

Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other. The main known usage of Ant is the build of Java applications. Ant supplies a number of built-in tasks allowing to compile, assemble, test and run Java applications. Ant can also be used effectively to build non Java applications, for instance C or C++ applications. More generally, Ant can be used to pilot any type of process which can be described in terms of targets and tasks.

Maven 3.2.1

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

GIT 2.8.1 64-bit

GIT is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

SourceTree 1.8.3

A free GIT & Mercurial client for Windows or Mac.

Oracle SQL Server 11g (11.2 xe)

Oracle Database 11g Express Edition (Oracle Database XE) is an entry-level, small-footprint database based on the Oracle Database 11g Release 2 code base. It's free to develop, deploy, and distribute; fast to download; and simple to administer.

Oracle SQL Developer 4.1.3 – 64 bit

Oracle SQL Developer is a free integrated development environment that simplifies the development and management of Oracle Database in both traditional and Cloud deployments. SQL Developer offers complete end-to-end development of your PL/SQL applications, a worksheet for running queries and scripts, a DBA console for managing the database, a reports interface, a complete data modeling solution, and a migration platform for moving your 3rd party databases to Oracle.

Tomcat Apache Server 8.x

The Apache Tomcat® software is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies. The Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket specifications are developed under the Java Community Process.

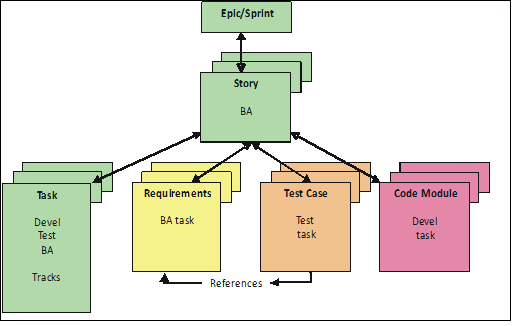
The Apache Tomcat software is developed in an open and participatory environment and released under the Apache License version 2. The Apache Tomcat project is intended to be a collaboration of the best-of-breed developers from around the world. We invite you to participate in this open development project.

Eclipse Mars 4.5.2

Eclipse provides IDEs and platforms nearly every language and architecture. We are famous for our Java IDE, C/C++, JavaScript and PHP IDEs built on extensible platforms for creating desktop, Web and cloud IDEs. These platforms deliver the most extensive collection of add-on tools available for software developers.

# Configuring and Using the FTL and JIRA Environment

This section describes the JIRA environment as it relates to the BCDSS effort. This section details the relationships between FTL, JIRA, epics, stories, tasks, requirements, test cases, and source code. The following figure depicts these relationships.



**Figure 2: BCDSS and the JIRA Environment**

A story contains all sprint artifacts and describes what the team implemented for the cycle. Although JIRA store artifacts, such as requirements and test cases, these items always link to a story in JIRA. In addition, source code check-in references the story.

The following sections describe the tasks associated with this effort based on where the event occurs in the development life cycle.

## Source Code Repository Overview

### Tool Source Control Overview

## Checking Files into Source Code Repository

### Refresh Remote Changes

### Refresh Sandbox

### Check-in and Deliver

### Resolving File Conflicts

### Configuring an External Compare Tool

# Development Guidelines

These sections provide guidance to developers to ensure a consistent approach for code artifacts.

## Implementing Front-End Components Standards

Adhere to the following standards when authoring front-end components:

* Controllers talk to Services instead of Data Access Objects (DAOs), whereas Services talk to DAOs and Delegates as needed.
* Access the BCDSS Database (BDB) via **Controller>Service>DAO**.
* Access the NoSQL Database (NDB) via **Controller>Service->Delegate>**.
* Scaffold code automatically styles all buttons (<button>) as jQuery buttons by default.
* Place script in <jsp:attribute name=”scripts”> instead of the <jsp:body> area.
* Use <label> for input-element field labels.
* Perform client-side validation for format/type.
* Leverage the showError(…) Java Script (JS) function to populate error messages from validation or server-side errors ().
  + ***Note***: Also use this function for showWarning() and showInfo().
* Some needed styles in BCDSS-app.css will evolve with UX assistance.
* Use Cascading Style Sheets (CSS) to lay out elements (versus HyperText Markup Language (HTML) tables).
* Use the basic rule for Tab Order of: Left-to-Right and Top-to-Bottom. Use the following HTML tabindex ranges to enforce keyboard Tab Order:
  + North Panel—0000-0499 range
  + West Panel—0500-0999 range
  + Center Panel—1000-1999 range (main content for screens)
  + East Panel—2000-2499 range
  + South Panel—2500-2999 range
* Use CSS class ‘priorityFocus’ for input elements that need default focus after page (*or partial page*) refresh. This allows the use of $(‘.priorityFocus’).focus() at any time.
* Ensure that you target Internet Explorer (IE) 11 for testing, which is the VA-supported browser.
* Use the NonVisual Display Access (NVDA) screen reader to smoke test accessibility concerns.
* openProgressBar and closeProgressBarare handled by global callbacks (*if using jQuery Ajax*).
* Use the $(‘#...’) jQuery notation instead of document.getElementById.
* When checking for error conditions from the server side, return error messages in WS payload; do not check data in with JS or create error messages in with JS, as this keeps JS minimal.

## Unsaved Changes Validation

## Required Fields Validation

The following sections cover the validation of the required fields.

### Description

BCDSS incorporates the jQuery ValidationEngine plugin to inform the user in a visual manner that the user failed to fill in required form fields. By default, validations range from email, phone, and URL, to more complex custom validation using AJAX processing or custom JavaScript functions. In the screen below, the Validation Engine visually warns the user of a failure to fill in the required fields for the page.

Figure 5: Example of Required Fields Errors

### Initialization

The Validation Engine requires the following files to enable:

jquery.js (jQuery)

validationEngine.js & validationEngine-en.js

validationEngine-jquery.css

Find the Validation Engine on the web at <https://github.com/posabsolute/jQuery-Validation-Engine>.

The Validation Engine initializes automatically for every page. The util.js function’s pageSetup function to call the enableFormValidation function to initiate the validation engine (this functionality requires no additional steps):

$("#form.id").validationEngine();

Prevent the Validation Engine from enableing by adding a hidden field in the page for pages with no necessary required field validation, as follows:

<input type="hidden" name=" skiprequiredfieldscheck">

### Applying Field Validation

This section provides the details for applying field validation. Find these code snippets in the file login.jsp. The following list provides examples of field validation:

1. When using it as CSS class type—Recommended for simple validation. It works best with simple text input fields:
   * class=”validate[required]”—Any field with this class requires this input control.
   * class=”validate[required, funcCall[validateUserIdType]]”—Calls required and custom function ‘validateUserIdType()’. Developers can hook custom functions. Refer to login.jsp and login.js.
2. When using it as Javascript—Recommended for more complex validation where a developer desires manual control over the display of an error prompt. In the case of input fields that dynamically disable or turn invisible, users must close the error prompt linked to that input field manually:
   * $(‘#input\_id’).validationEngine(‘hidePrompt’); //if you want to remove error message manually.
   * $('#input\_id').validationEngine('showPrompt', 'This a custom msg', 'load');//3rd argument is for background color: ‘load’ = black, ‘pass’=green and anything else = red.
   * $(‘#form\_id’).validationEngine(‘hide’);//hide all errors for form.
   * alert($(‘#form\_id’).validationEngine('validateField',‘#input\_id’) );//validates only one field.
   * $("#form\_id").validationEngine("updatePromptsPosition"); //update prompt’s position. Useful when re-positioning of prompt is necessary.
3. Pre-defined validators:
   * required: Fails if the element contains no value. This validator can apply to almost any kind of input field.

**Sample usage:**

<input value="" class="validate[required]" type="text" name="email" id="email" />

<input class="validate[required]" type="checkbox" id="agree" name="agree"/>

<select name="sport" id="sport" class="validate[required]" id="sport">

<option value="">Choose a sport</option>

<option value="option1">Tennis</option>

</select>

* + custom[regex\_name]: Validates the element's value to a predefined list of regular expressions.

**Sample usage:**

<input value="someone@nowhere.com" class="validate[required,custom[email]]" type="text" name="email" id="email"/>

* + funcCall[method\_name]: Validates a field using a third party function call. If a validation error occurs, the function must return an error message that will automatically show in the error prompt.

**Sample usage:**

function checkHELLO(field, rules, i, options){

if (field.val() != "HELLO") {

// this allows the use of i18 for the error msgs

return ‘Text is not HELLO. Please enter HELLO’;

}

}

<input class=”validate[required, funcCall[checkHELLO]]” type=”text” id=”hello” name=”hello”/>

* + equals[input\_id]: Check if the current field's value equals the value of the specified field.

### Usage Tips

Use the following tips for the validation engine:

* The page must contain a unique field.id across the page.
* For simplicity and consistency, field.id and field.name should match (except with minCheckbox and maxCheckbox validators.
* Spaces and special characters should be avoided in field.id or field.name.
* Use lower case for input.type; e.g., text, password, textarea, checkbox, radio.
* Validators evaluate from left to right. Use the Ajax validator last e.g.: validate[custom[onlyLetter],length[0,100],ajax[ajaxNameCall]].
* Use only one Ajax validator per field.

## Miscellaneous Notes

This section contains miscellaneous information important to know for maintaining development environments.

### Target Screen Resolution

Mobile devices notwithstanding, plan the target screen resolution for BCDSS as 1024x768 using a 4:3 monitor on a desktop/laptop machine using the approved version of Internet Explorer with Text Size of “Normal.” Developers and testers must ensure no horizontal scrolling against this configuration.

1. Terminology

The following index lists the terminology used in this document.

| **Acronym** | **Definition** |
| --- | --- |
|  |  |

1. References

The following documents supplement the information in this document:

* 1. User Documentation

1. Approval Signatures

This section is used to document the approval of the BCDSS Technical Manual during the Formal Review. The review should be conducted face to face where signatures can be obtained ‘live’ during the review. If unable to conduct a face-to-face meeting then it should be held via LiveMeeting and concurrence captured during the meeting. The Scribe should add /es/name by each position cited.

The Business Sponsor and Project Manager are required to sign.

REVIEW DATE:

SCRIBE:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signed: Date:

< Business Sponsor >

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signed: Date:

< Project Manager >