

AgileSites 0.5

Developer Guide

A *modern* framework to build **easily** great sites
with Oracle WebCenter Sites

By Michele Sciabarra
<Michele@sciabarra.com>



This document is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License](#).

Software is Released under Apache LICENSE 2.0
(see LICENSE file)

NOTE. This is a work-in-progress version of the AgileSites developer guide. It is still incomplete and it can omit information, or contain incorrect or obsolete information.

1. Introducing AgileSites

1.1. How to read this book

This book is a complete introduction to WCS site development with AgileSites. This book is written for developers and covers how to implement a web site template in WebCenter Sites using AgileSites.

This book is **not** written for web designers, so it does not cover how to create html, css, javascript for your web site. Actually, it starts assuming you already have an HTML mockup of your website (usually provided by a different team).

You should start with **Chapter 1** - Introducing AgileSites to learn what is and what can do for you. It does not contain technical information, though, so you may skip it if you are not interested.

You should check Chapter 2 - Developing with AgileSites to learn how to install it and start developing with it.

Chapter 3 - Site Design for WCS is probably the most important chapter of the book, It introduces best practices and a methodology for site design for AgileSites.

<p>NOTE. You should read chapter 3 very carefully, more than once if necessary. It contains key concepts that are the pillars on which both AgileSites and WCS itself is built.</p>
--

After you understood this chapter, you can go on and design your content model. Once done, you can finally start implementing it. If you are a Java coder, **Chapter 4** - Rendering in Java with AgileSites will teach you how to implement the mockup in Java.

If you are a Scala programmer, or you are willing to learn and use Scala for your site, then **Chapter** Error! Reference source not

found. - Error! Reference source not found. will also teach the API wrapper to render the site using Scala.

The biggest advantage of using Scala is simpler code written in a functional style that is usually shorter. Scala is a language that builds on top of Java, adding a number of interesting features, most of them based on the functional programming. Generally, a Java developer can learn enough of Scala to be able to use the framework in just few days, gaining a lot in conciseness, expressivity and power.

1.2. Prerequisites

Developers are required to know before starting to use:

- a good working knowledge of the Java programming language
- knowledge of HTML, CSS and Javascript, be able to read and perform some modification to client code
- how to use the WebCenter Sites as an user (described in the WCS User Manual)
- how to define a content model for Web Center Sites (described in the WCS Developer Manual)

<p>NOTE. It is planned for a future version of this book, a chapter how to design the content model, in order to make this book standalone for developers.</p>

Developers are not required to know the WCS API to perform standard tasks since AgileSites api wraps it in a way that tries to simplify development and enforce best practices.

However, AgileSites allows to access to the full WCS API, so developers interested in performing more advanced tasks can learn WCS api and still use it directly from AgileSites.

In short, AgileSites offers gentle path to start developing with WCS with a simpler API that conforms to best practices, and

then allows you to use the full WCS API for more experienced developers.

1.3. What is Oracle WebCenter Sites

Oracle WebCenter Sites (in short, Sites) is a Web Experience Management system owned and licensed by Oracle Corporation. However, WEM is the term used to define a Content Management System specifically oriented to build websites (to distinguish from document management that is also considered content).

WCS is the result of an acquisition, it was previously known as Fatwire ContentServer. Before Fatwire, the product was known as Divine ContentServer, OpenMarket ContentServer and FutureTense ContentServer.

It has a very long story of development and it is used by a number of large companies including, according to an interview to the former Fatwire CEO, Yogesh Gupta, Apple and 3M.

1.4. What is AgileSites

AgileSites is a framework for Oracle WebCenter Sites that is the result of more than ten years of experience of the author on the platform.

It improves the developer experience offering a number of interesting features:

- It tries to be 100% compatible with WCS, using only documented API.
- It follows best practices, coding them in the framework itself and in the documentation, providing guidelines to avoid mistakes and get results quickly.
- It allows site coding using plain Java, without any knowledge of the JSP.
- No need to restart the application server to pickup changes when you use Java, still you can use pure Java or Scala coding through a class loader.

- A simplified API in Java and Scala is available to make development easier. Scala offers an embedded DSL to make development even easier.
- The full standard WCS API is still accessible through tag wrappers or directly using Asset API.
- No special plugins are needed to edit code, just use Eclipse or any other IDE supporting Java (or Scala if you use it).
- A build system is part of the framework. Just get the framework, import in eclipse and you can start developing with it.
- Deployment is greatly simplified: a single jar including all the code, that can be easily tracked and distributed.

More information about the reasons because AgileSites was developed can be found in the blog www.sciabarra.com/fatwire

2. Developing with AgileSites

NOTE. Current version (0.5) supports only development and deployment on fatwire installation present in the same machine. So you need to develop either using the Jump Start Kit or a local install of the product. Version 0.4 will introduce sremote development and publishable resources.

2.1. Installation

This section describes quickly how to install AgileSites in WCS and start developing. More details on the specific steps are provided in the following paragraph in this same chapter.

2.1.1. Download

AgileSites is available from github and is distributed as a zip or tar.gz. It can also be downloaded in source format but it is recommended to download a stable version in source format. The framework itself is distributed in source format and not in binary format.

2.1.2. Prerequisites and Configuration

1. Install either Oracle WebCenter Sites full version or JumpStartKit. Framework can be compiled for 7.6 but samples require at least 11gR1.
2. Download a stable release of AgileSites of the version this book refers to (check the first page).
3. Copy `build.sbt.dist` in `build.sbt`
4. Configure `build.sbt`. Comments should guide you what to do, more details are provided in the **Configuration** paragraph later in this chapter.

2.1.3. Offline setup

5. Ensure WCS is **not running**.
6. Start the AgileSites command line shell (run either `AgileSites.sh` on Unix (Linux/Mac) or `AgileSites.bat` on Windows)
7. Type in the shell (note that the first step will take some time and requires an active internet connection):

```
core/clean
core/publish-local
wcs-setup
```

2.1.4. Online setup

8. Start WCS, and ensure it is **up and running**.
9. Import the main entry point with the following command:

```
wcs-cm import_all
```

2.1.5. Importing samples

In this paragraph we import the 2 sample sites provided with the framework.

NOTE. When your site is deployed, you will instead follow this procedure to import your site instead. In the following chapters we will guide step-by-step to build your sites. You will then change only the **wcsSites** configuration for importing yours but the procedure is the same.

- 10.
11. Execute the following csdt commands to import samples or your site.

```
wcs-dt import @SITE
wcs-dt import @ASSET_TYPE
wcs-dt import @ALL_ASSETS
wcs-dt import @STARTMENU
wcs-dt import @TREETAB
```

12. Package and deploy your code

`wcs-deploy`

13. Build eclipse configuration files with

`eclipse`

and then import the projects in eclipse. You will see 3 new projects: `agilwcs-core`, `AgileSites-app` and `agiulewcs-api`.

14. Access the admin site and assign to users of the sites appropriate roles. For example in development environment I normally assign to the *fwadmin* users the roles **AdvancedUser**, **SitesUser** and **GeneralAdmin**

TODO: Step by step description how to assign roles.

2.2. Testing

TODO: explain what to test here

Congratulations! You are done! You can now start coding. Check the Development paragraph in this chapter for details.

But before, please learn more about designing your content model and your rendering process.

2.3. Configuration

In this paragraph we provide detailed information about the configuration file of the framework.

NOTE. Separate settings with a blank line and **do not add a `;`** to the end. Comments here starts with `//` not with `#`.

You should start copying `build.sbt.dist` in `build.sbt`, then edit `build.sbt` and set:

1. `wcsSites` in `ThisBuild`
to the sites you are currently working on with AgileSites. It is a comma-separated list of site names.
2. `wcsVersion` in `ThisBuild`
to the version of your Fatwire/WCS. Currently 7.6 or 11g.
IMPORTANT: sample sites cannot be imported in 7.6.
3. `wcsHome` in `ThisBuild`
to the home WebCenter Sites application directory
4. `wcsShared` in `ThisBuild`
to the WebCenter Sites shared folder
5. `wcsCsWebapp` in `ThisBuild`
to the web application folder of the Content Server application (usually named **cs**)
6. `wcsSsWebapp` in `ThisBuild`
to the web application folder of the Satellite Server application (usually named **ss**)
7. `wcsCsdtJar` in `ThisBuild`
to the location of your CSDT jar.
Position of this file may vary, in JSK it is located in `wcsHome` folder, in full installations it is usually stored in a subdirectory.
8. `wcsUrl` in `ThisBuild`
to the url to access content server.
Please DO NOT include a slash at the end.
Include the protocol (`http` or `https`), the port and the path to ContentServer (for example `/cs`). This url must expose Content Server servlets (**ContentServer**, **CatalogManager**)
9. `wcsUser` in `ThisBuild`
is the username of an user with developers privileges for your sites
10. `wcsPassword` in `ThisBuild`
is the password of the `wcsUser`.
The following settings requires a reinstallation if changed since those settings are stored in the code.
11. `wcsFrontUrls` in `ThisBuild`
is a sequence of associations site to the entry point url for each website. See TODO for a discussion of the possible values.

12. `wcsFlexBlobs` in `ThisBuild`

Regular expression to match blobs stored in Flex Attributes –matching a numeric id is required to locate a flex blob.

It is important to tweak the extensions matched to include those used in the site. See TODO for a discussion of the subject.

13. `wcsStaticBlobs` in `ThisBuild`

Regular expression to match blobs stored as static assets.

Their name generally matches the filename of the static asset itself. It is important to tweak the extensions matched to include those used in the site. See TODO for a discussion of the subject

2.3.1. Example Configuration

TODO insert here

2.4. Eclipse

Once AgileSites is installed, 3 project files are generated. To use them, you need Eclipse or another IDE (like NetBeans or IntelliJ) that can import Eclipse project file

Let's focus now only on Eclipse. Only basic Java editing capabilities are required, so you can use any version of Eclipse used as long as it supports Java project.

However, since coding a website may involve editing of HTML, CSS and JavaScript files, Eclipse for J2EE development is recommended.

Furthermore, if you plan to do also development in Scala (that is not mandatory, since a full Java API is provided), you must install the Scala-IDE Eclipse plugin (<http://www.scala-ide.org>).

Last but not least, to avoid running the command line shell outside of windows, the local terminal plugin is recommended, so overall the minimum Eclipse configuration recommended is:

- Eclipse Juno (or newer) for JEE Developers
- Scala IDE plugin for Scala 2.10.0 (or newer)
- Local Terminal plugin

14.

NOTE. I do not recommend (yet) installing the CSDT eclipse plugin because in my experience it does not work with Eclipse Juno.

2.4.1. Configuring Local Terminal

To import projects generated by the AgileSites shell, you have to go to the menu **File | Import** then under General select **Existing Projects in the Workspace** (Figure 1). Then select the top directory of your AgileSites installation. You should see the 3 standard AgileSites subprojects :

- AgileSites-app,
- AgileSites-api
- AgileSites-core

Select all of them, click on finish and now you can start coding.

NOTE. Strictly speaking, only the **AgileSites-app** project is needed. The **AgileSites-api** project contains library code, basically the layer on top of WCS API that is there to make development easier. You should write all your code in the **AgileSites-app** project. You should place reusable library code in the **AgileSites-api** project. Except for developing the framework itself, you should never change the **AgileSites-core** (and if do, please consider contributing your changes to the project).

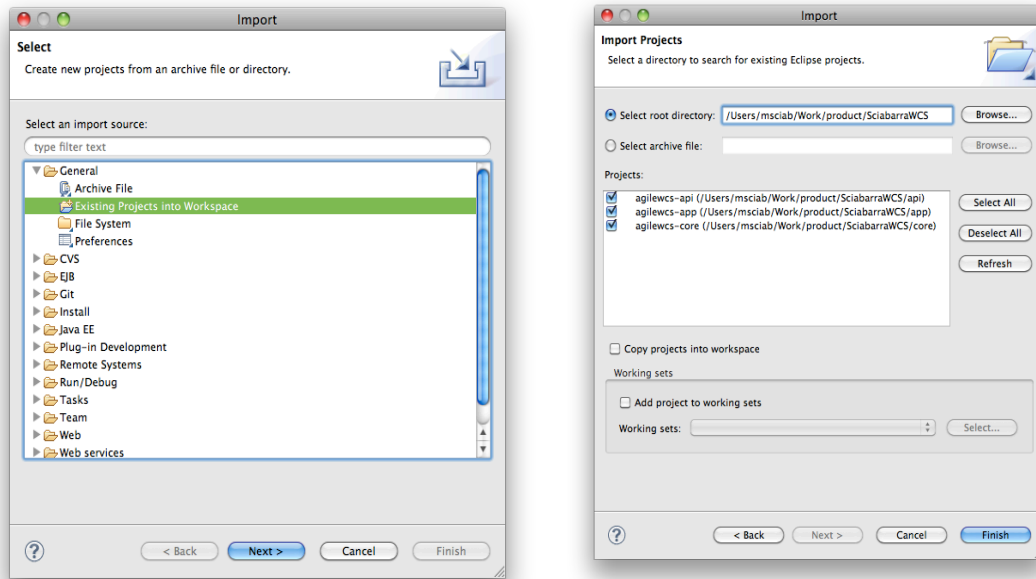


Figure 1 Importing projects in Eclipse

2.5. AgileSites shell

The AgileSites shell is actually (it is not a secret) a customized SBT (Simple Build Tool) project. SBT is a configurable build tool, used mostly for Scala programs but able to build Java programs, too.

Since it is expandable and provides an interactive interface, it was used as the foundation for the AgileSites shell used for the framework, working fine also for Java developers.

2.5.1. Continuous deployment

After you performed the installation, started the application server and the AgileSites shell, there is only one command you need to learn:

```
~ app/wcs-deploy
```

This command will build and deploy the AgileSites application continuously, while you write code. Any content editing will trigger deployment.

However, when you are not editing css and javascript files and do not need to change Setup classes, you can shorten the process and just run

```
~ app/wcs-package-jar
```

Also this command is continuous and any code change will trigger deployment. Probably 90% of the coding time is spent using this command active.

To better understand the syntax, the actual command is **wcs-deploy**. The command **app/wcs-deploy** will perform the command only for the subproject app, and **~ app/wcs-deploy** enable continuous deployment: the command is executed again when any of the source files change.

2.5.2. The **wcs-deploy** dissected

The command **wcs-deploy** actually perform 3 different subcommands, that can be invoked individually:

- **wcs-update-asset**
- **wcs-copy-static**
- **wcs-package-jar**

The command **wcs-update-asset** will create or update structural assets created by the framework. Invoking it is required only when the Setup classes of a site change (see next chapters to read about those classes).

The command **wcs-copy-static** copies into a folder accessible by the CMS the static content (normally javascript and css files).

The command **wcs-package-jar** is the key command. It compiles source code, and also package html files (but not javascript and css) from the static folder and deploy it to a location monitored by the core of the framework. As a result, classes are immediately reloaded and the developer can test the result of the change without any need to restart the application server.

Since changing html and source code is usually done 95% of the time, a developer may want to keep only **wcs-package-jar** in

continuous deployment mode, and occasionally execute manually **wcs-copy-static** or **wcs-update-assets**.

2.6. CSDT

A CSDT client (Content Server Development Tool) is integrated in the AgileSites shell.

Setup will configure csdt to use the folder `export/envision` inside the project.

It is restricted to use only sites with user and password configured in the `build.sbt`.

The command to use csdt in the shell is `wcs-dt` followed by two arguments, a command and a *SELECTOR* (described below).

- `wcs-dt listcs SELECTOR`
`wcs-dt listds SELECTOR`
returns a list of selected assets in the current sites in cs (no universal id) or ds (with universal uid).
- `wcs-csdt export SELECTOR`
exports selected assets the current sites
- `wcs-csdt import SELECTOR`
imports selected assets in the current sites
- `wcs-dt`
is short for `wcs-dt listds @ALL_ASSETS`

2.6.1. Selectors

Please check “WCS Developer Tools” for detailed informations. Here there are some samples of the syntax:

- `@ALL_ASSETS` all the assets
- `@ALL_NONASSETS` all the non-assets (startmenu, treetab. assets types)
- `@TREETAB` all the treetabs
- `@STARTMENU` all the start menus
- `@ASSET_TYPE` all the asset types

- Page all the Page (you can use any other asset type)
- `xxx:123456` the asset type `XXX` and id `123456` (can be any asset or a non-asset with an id)
- `xxx:12345,123456` the assets of the given ids.

2.6.2. Catalog Manager

Catalog Manager command line is integrated in the AgileSites Shell.

TODO

3. Site Design for WCS

Proper design both of your Content Model and Rendering Process is of paramount importance for the success of your WCS project. Without a good design, your implementation will suffer of many problems. For example:

15. Incorrect or inefficient Content Model will make painful and inefficient for content to build the content for the site
16. Lack of design for the rendering process will make the code confused, repetitive, and what is worse buggy and much more complex than necessary.

In this chapter we introduce a design procedure that is valid for any Fatwire/WCS implementation. Implementation could be done with plain JSP templates, and the methodology herein described is still valid.

However AgileSites framework, since has been designed by the same author of this process, enforcing this design process, providing out-of-the-box straightforward api to implement the design in code, and a simplified API that wraps many of the complexities of traditional WCS JSP coding.

.

3.1. *Identify your layouts*

It is assumed an html mockup of your site is available. This book does not cover html design, only how to implement the html design in a content managed web site.

Each site has a number of layouts. They are HTML files that implement a full html .

For each layout you have to create a Template (see the API) of type Layout

3.2. Create the site

TODO

3.3. Create the Content Model

TODO

4. Rendering in Java with AgileSites

This chapter explains how to content manage an HTML template. It assumed you already designed your site structure, decided for your content types and created a hierarchy of Templates and CSElements.

Inizi renderizzando un articolo

Se vai sulla contributor interface, content tree, Articles e poi SideBlock vedi due articoli, Welcome e Author.

Clicca su Welcome, e poi clicca sullo switch "form" per farlo diventare "web"

Dovrebbe apparire "hello world";

A questo punto puoi cominciare

Vai su `AgileSites.article.Layout`

La classe sara'

```
public class Layout extends Element {  
    @Override  
    public String apply(Env e) {
```

```
        return "hello world";  
    }  
}
```

Quello che succede e' che eseguendo il preview dell'articolo viene chiamato il template Layout per Agile_Article (il tipo del contenuto) che comincia a renderizzarlo. Siccome il corpo ritorna una stringa questo e' tutto quello che vedi.

Ora comincia a estendere il rendering. Prima di tutto metti

```
Asset a = e.getAsset();  
return a.getName();
```

In questo modo prendi l'asset corrente e lo leggi.

Switcha a form e poi a web di nuovo (e dovresti avere ~ wcs-package-jar in esecuzione) e dovresti vedere il nome dell'articolo.

Puoi provare a fare:

```
Long id = a.getId();  
String name = a.getName();  
String desc = a.getDescription();  
Date date = a.getStartDate();
```

per leggere i vari campi che sono comuni a tutti gli asset

E puoi usare invece

```
String author = a.getString("Author");  
Date today = a.getDate("Birthday");  
Integer amount = a.getInt("Count");
```

per leggere gli attributi come stringa, data o intero (e dipendono dagli attributi definiti per il contenuto corrente)

Ora prova a utilizzare invece l'html.

L'idea è che tu prenda un html pulito e operi delle sostituzioni per renderizzare il template.

Fai:

```
Picker p = new Picker("/index.html");  
return p.html();
```

Se fai così lui prende l'html da `src/main/resources/index.html` (il path è assoluto ma la radice è `resources`) e lo stampa. Se fai

preview dovresti vedere l'html ma senza stili, perche' gli stili sono configurati per funzionare con il browser in locale

La prima cosa che devi fare e' aggiustare il path dei css

Fai:

```
Picker p = new Picker("/index.html");  
p.attr("head link", "href", "/cs/css/default.css");  
return p.html();
```

questo cambia l'attributo href del link nella head a "/cs/css/default.css", se fai preview dovresti vedere il css aggiustato.

La "head link" e' un selettore stile jquery. Qui ci sono i selettori disponibili:

<http://jsoup.org/cookbook/extracting-data/selector-syntax>

Quindi ora puoi cominciare a mettere le cose insieme.

```
Picker p = new Picker("/index.html");  
p.attr("head link", "href", "/cs/css/default.css");  
p.replace("div#content h1.title", a.getName());
```

```
        p.replace("div#content    p.meta    small", "by " +  
a.getString("Author"));  
        return p.html()
```

4.1. Implementing the Article

In this section we will see how to implement the Article, defining the content type and then

4.1.1. Creating the Article content type

Let's start from the Article.

Create a new

4.2. Configure

4.3. Create template

4.4. Create your site
