AgileWCS Developer Guide

*Everything you need to know to **easily** build great sites for Oracle WebCenterSites with AgileWCS.*

Author:	Michele Sciabarra michele@sciabarra.com
Version:	AgileWCS AgileWCS 0.3
Status:	draft

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

1. Introducing AgileWCS

1.1. How to read this book

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

It 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 **Chapter1 – Introducing AgileWCS** 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 AgileWCS** to learn how to install it and start developing with it.

Chapter 3 – Site Design with WCS is probably the most important chapter of the book, and is **not** AgileWCS specific. It introduces best practices and a methodology for site design that is good both for plain WCS and AgileWCS.

NOTE. You should read chapter 3 very carefully, more than once if necessary. It contains key concept that are the pillars on which both AgileWCS and WCS itself is built.

After you understood Chapter 3, 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 AgileWCS** 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 5 – Rendering in Scala with AgileWCS** 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. However Scala is a different language that must be learnt (although a basic knowledge of Scala is enough).

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)

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.

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

However, AgileWCS 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 AgileWCS.

In short, AgileWCS 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 WCS

Oracle WebCenter Sites (in short, WCS or just Sites) is a Content Magement System owned and licensed by Oracle Corporation. Currently, Oracle defines it a Web Experience Management system.

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 AgileWCS

AgileWCS 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.
- If 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 or Scala, it addition to JSP. Scala is completely optional and only Java coding skills are actually needed.
- 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 AgileWCS was developed can be found in the blog www.sciabarra.com/fatwire

2. Developing with AgileWCS

NOTE. Current version (0.3) 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 AgileWCS 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

ScalaWCS is available from github and is distribuited as a zip or tar.gz. It can also be downloaded in source format but it is recommended to downlad a stable version in source format. The framework itself is distribuited 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 AgileWCS 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 agilewcs command line shell (run either `agilewcs.sh` on Unix (Linux/Mac) or `agilewcs.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 yoior 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 @ALL_NONASSETS
```

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, agilewcs-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.

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

1. `wcsHome in ThisBuild`
   to the home WebCenter Sites application directory
2. `wcsWebapp in ThisBuild`
   to the web application folder
3. `wcsVersion in ThisBuild`
   to the version of your Fatwire/WCS. Currently 7.6 or 11g
4. `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 may be in a subdirectory.

5. `wcsSites in ThisBuild`
   to the sites you are currently working on with AgileWCS. It is a comma-separated list of site names.
6. `wcsUrl in ThisBuild`
   to the url to access content server.
   Please include a slash at the end.
   Include the protocol (`http` or `https`), the port and the path to ContentServer (for example `/cs`)
7. `wcsUser in ThisBuild`
   is the username of an user with developers privileges for your sites
8. `wcsPassword in ThisBuild`
   is the password of the wcsUser.
9. `wcsFront in ThisBuild`
   TODO

15.

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

2.3.1. Example Configuration

```
// versions (currently only valid values are 11g or 7.6)
wcsVersion in ThisBuild := "11g"

// home directory of WCS
wcsHome in ThisBuild := "/JSK/11.1.1.6.1/Sites/11.1.1.6.1/"

// webapp directory of WCS
wcsWebapp in ThisBuild := "/JSK/11.1.1.6.1/App_Server/apache-
tomcat 6.0.32/webapps/cs/"

// location of the csdt-client jar
wcsCsdtJar in ThisBuild := "/JSK/11.1.1.6.1/Sites/11.1.1.6.1/csdt-
client-1.2.2.jar"

//url to content server
wcsUrl in ThisBuild := http://localhost:8080/cs/

// sites to import/export
wcsSites in ThisBuild := "AgileWCS,ScalaWCS"

// user to import/export
wcsUser in ThisBuild := "fwadmin"

// password to import/export
wcsPassword in ThisBuild := "xceladmin"

// frontend url – slash terminated
wcsFrontUrl in ThisBuild := "http://localhost:8080/cs/Satellite/”
```

2.4. Eclipse

Once AgileWCS 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

16.

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

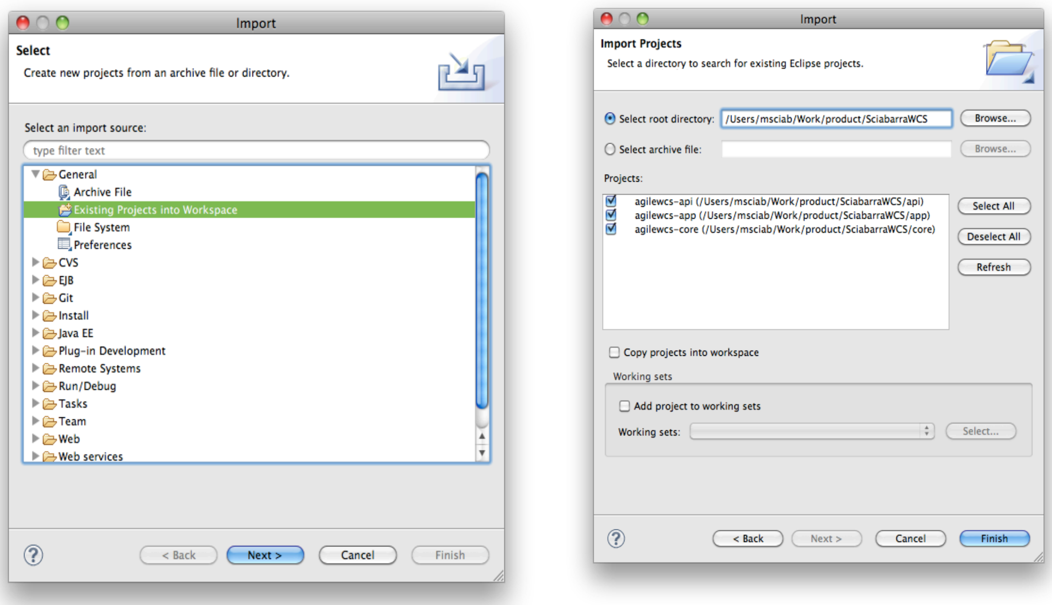
2.4.1. Configuring Local Terminal

To import projects generated by the agilewcs 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 AgileWCS installation. You should the 3 standard AgileWCS subprojects :

- agilewcs-app,
- agilewcs-api
- agilewcs-core

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

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



Select root directory: /Users/msciab/Work/product/SciabarraWCS
Select archive file:
Projects:
- [✓] agilewcs-api (/Users/msciab/Work/product/SciabarraWCS/api)
- [✓] agilewcs-app (/Users/msciab/Work/product/SciabarraWCS/app)
- [✓] agilewcs-core (/Users/msciab/Work/product/SciabarraWCS/core)

[ ] Copy projects into workspace
Working sets
[ ] Add project to working sets
Working sets:

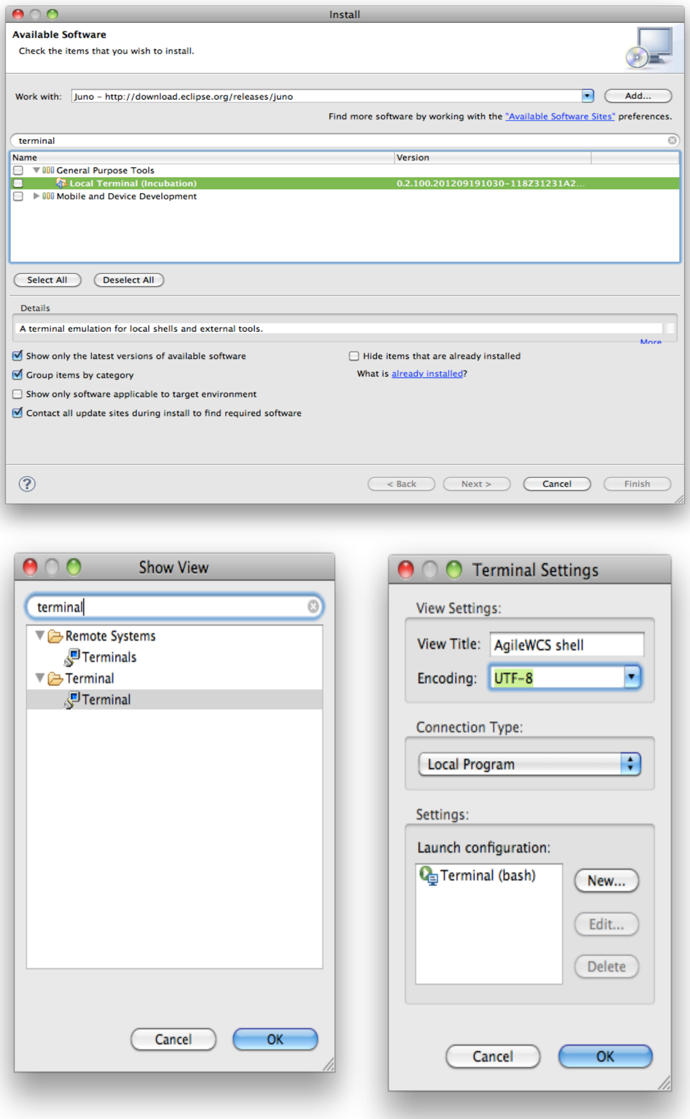
Figure 1 Importing projects in Eclipse

2.4.2. Configuring the local terminal

AgileWCS relies on its command line shell to perform a number of tasks. This paragraph describes how to install a local terminal plugin for Eclipse to run the shell, in order to get a truly integrated environment.

NOTE. Using the local terminal inside eclipse is not mandatory. Indeed, it is currently a bit buggy, so you may use a separate terminal window.

Local terminal is not currently part of Eclipse Juno but you can install it from **Help | Install New Software**, then selecting **Juno** download, search for *terminal* and finally select *Local Terminal* (Figure 2). Then check the checkbox and complete install. You have now available a new view that can execute a local shell in your Eclipse; to use it got to **Window | Show View | Other**, search for *terminal* and select the icon *Terminal* under the folder *Terminal* (not *Terminals* under *Remote Systems*). The system will then ask you to create a new run configuration.


Install

Available Software
Check the items that you wish to install.

Work with: Juno – http://download.eclipse.org/releases/juno

Find more software by working with the "Available Software Sites" preferences.

terminal

Name	Version
☐ ▼ General Purpose Tools	
☐ Local Terminal (Incubation)	0.2.100.201209191030-118Z31231A2...
☐ ▶ Mobile and Device Development	

Details
A terminal emulation for local shells and external tools.

☑ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software
☐ Hide items that are already installed
What is already installed?

Show View

terminal
- Remote Systems
  - Terminals
- Terminal
  - Terminal

Terminal Settings

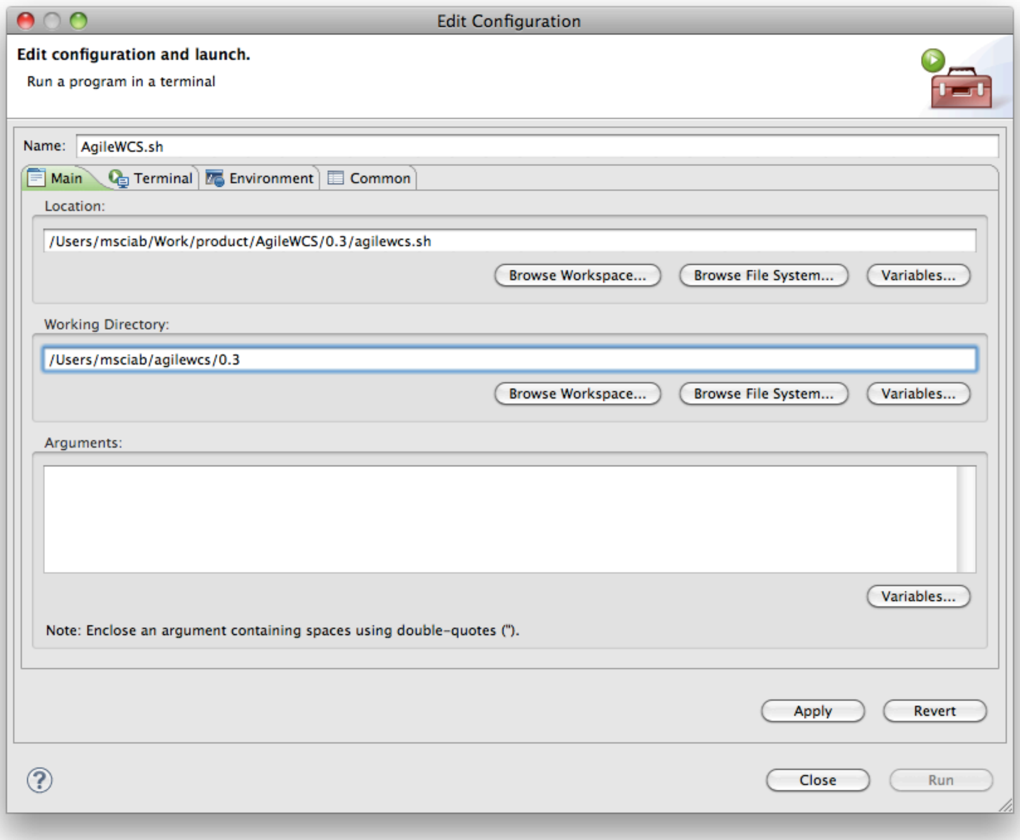
View Settings:
View Title: AgileWCS shell
Encoding: UTF-8

Connection Type:
Local Program

Settings:
Launch configuration:
Terminal (bash)

Figure 2 Installing the local terminal

You can now create the local terminal configuration (Figure 3). Simply select the **agilewcs.sh** or **agilewcs.bat** script and run it. Yo will get a frame inside the main Eclipse window where you can interact with the shell. Check next paragraph to read what you can do with the AgileWCS shell.



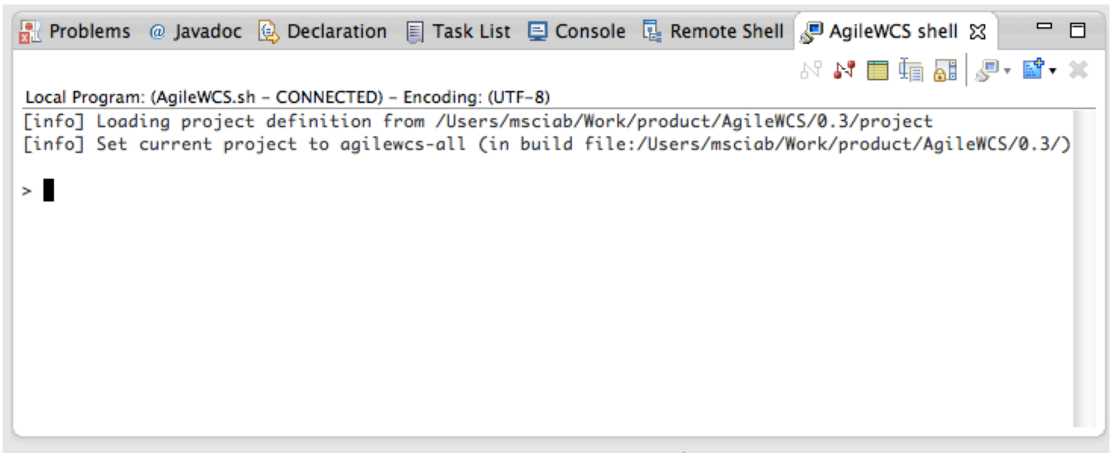
Name: AgileWCS.sh

Location:
/Users/msciab/Work/product/AgileWCS/0.3/agilewcs.sh

Working Directory:
/Users/msciab/agilewcs/0.3

Arguments:

Note: Enclose an argument containing spaces using double-quotes (").



```
Local Program: (AgileWCS.sh - CONNECTED) - Encoding: (UTF-8)
[info] Loading project definition from /Users/msciab/Work/product/AgileWCS/0.3/project
[info] Set current project to agilewcs-all (in build file:/Users/msciab/Work/product/AgileWCS/0.3/)

>
```

Figure 3 Running local terminal

2.5. AgileWCS shell

The AgileWCS 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 AgileWCS 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 agilewcs shell, there is only one command you need to learn:

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

This command will build and deploy the agilewcs 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 be 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.**

2.5.3. CSDT

(TODO - INCOMPLETE PARAGRAPH)

The CSDT client (Content Server Development Tool) is integrated in the AgileWCS shell.

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

The command to use csdt is ``wcs-csdt`` followed by one or more arguments.

- ``wcs-csdt`` returns a list of assets in the selected site

- ``wcs-csdt export`` exports all the assets from the current site

- ``wcs-csdt import`` imports all the assets in the current site

# Other examples

The correct synopsis is ``wcs-csdt command selection``, where:

Command defaults to ``listcs`` but can be

- ``listcs``

- ``listds``
- ``export``
- ``import``

Selection defaults to ``@ALL_NONASSETS;@ALL_ASSETS`` but can be any syntax supported by CSDT.

See the WCS Developer Tools documentation for more details.

Exporting Assets with csdt

The shell integrates CSDT and provides a simplified interface to invoke.

Here there are the relevant commands:

<pre>

wcs-dt

</pre>

This will list the assets in the selected sites in the built.sbt

This will list all the assets and non assets in the selected sites

<pre>

wcs-dt listcs [_SELECTION_]

wcs-dt listds [_SELECTION_]

</pre>

This will listcs or listds the entities specified as _SELECTION_ (default: all the assets and the non assets).

Check the WCS Developer Tools manual for an explanation of the available syntax.

See below for a quick reference of the selection syntax

<pre>

wcs-dt export _SELECTION_

wcs-dt import _SELECTION_

</pre>

This will import or export assets and non assets specified in the selection

# SELECTIONS

Some samples:

- @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 entity of the given type XXX and given id (can be an asset or a non-asset)

- XXX:12345,123456 the entities of the given ids.

2.5.4. Catalog Manager

(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:

- Incorrect or inefficient Content Model will make painful and inefficient for content to build the content for the site
- 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 AgileWCS 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. Create the site

3.2. Create the Content Model

4. Note

4.1. 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

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 agilewcs.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 contento) 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 contentuo corrente)

Ora prova a utilizzare invece l'html.

L'idea e' che tu prendi un html pulito e operi delle sostiuzioni per renderizzare il template.

Fai:

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

Se fai cosi' lui prende l'html da src/main/resources/index.html (il path e' assoluto ma la radice e' resources) e lo stampa. Se fai preview dovresti vedere l'html ma senza stili, perche' gli stilo sono configurati per funzioanare 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()
```

5. Rendering in Java with AgileWCS

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.

5.1. Implementing the Article

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

5.1.1. Creating the Article content type

Let’s start from the Article.

Create a new

5.2. Configure

5.3. Create template

5.4. Create your site

6. Rendering in Scala with AgileWCS

6.1. # API

Dispatcher invokes classes implementing the ``wcs.Element`` trait.

Each Element must implement the method ``apply(e: Env)``

This ``x: Env`` variable is the entry point to invoke the enviroment.

## Accessing variables and lists ##

- Read a variable as a String: <br>``e("var")``

- Read a list as a sequence and then iterate it: <br>``e.list("list")``

Example:

<pre>

for(row &lt;- x.list("list")) {

   row("value")

}

</pre>

6.2. ## Getting the current asset

(TODO)

- Retrieve an asset using current c/cid: <br>``val a = x.asset``

- Retrieve the id of the current asset using c/cid:<br>``val id = x.id``

- Retrieve the id of the asset using different variables:<br>``val id = x.id("otype", "oid")``

- Create a new asset Id:<br>``Id("Page" x("otype"))``

- Retrieve an asset using different variable values:<br>``val a = x.asset(x.id("otype", "oid"))``

6.3. ## Calling another Template ##

(TODO)

- Call a template with current c/cid:<br>``x.call("Body")``

- Call a template with different id value:<br>``x.call(x.id("otype","oid"), "Body")``

6.4. ## Get Template Url

(TODO)

- Generate a url calling the given template (default Layout):<br>``x.url(x.id)``

6.5. ## Accessing attributes

(TODO)

Given ``a: Asset = x.asset``

- Retrieve an attribute (single value):<br>``a("attribute")``

- Retrieve a group of attributes:<br>``(title, summary) = a("Title", "Summary")``

- Iterate the multiple values of a multivalued attribute:<br>

<pre>

&lt;ul>{

    a("children") {

      (title, related) =>

       &lt;li>&lt;a href={url(x("c"), related)}>{title}&lt;/a>&lt;/li>

    }

}&lt;/ul>

</pre>

7. AgileWCS Architecture and Internals