Oracle® WebCenter Sites

Administrator's Guide for the Site Capture Application

11g Release 1 (11.1.1)

April 2012

ORACLE®

Oracle® WebCenter Sites: Administrator's Guide for the Site Capture Application, 11*g* Release 1 (11.1.1)

Primary Author: Tatiana Kolubayev

Contributor: Amit Kumar, Kamal Kapur

Table of

Contents

About This Guide

This guide describes Oracle WebCenter Sites: Site Capture, a web application that is used to download dynamically published websites.

This guide begins with an overview of the Site Capture application. It shows you how to navigate the Site Capture application and its file system to manage crawl sessions and downloaded websites. The concluding chapter presents configuration code used to control the crawler's site capture process. Examples illustrate the use of Java methods and interfaces for implementing link extraction logic, rewriting URLs, enabling email notification at the end of a crawl session, and otherwise customizing the capture process.

Applications discussed in this guide are former FatWire products. Naming conventions are the following:

- *Oracle WebCenter Sites* is the current name of the application previously known as *FatWire Content Server*. In this guide, *Oracle WebCenter Sites* is also called *WebCenter Sites*.
- *Oracle WebCenter Sites: Site Capture* is the current name of the application previously known as *FatWire Site Capture.* In this guide, *Oracle WebCenter Sites: Site Capture* is also called *Site Capture.*
- *Oracle WebCenter Sites: Web Experience Management Framework* is the current name of the environment previously known as *FatWire Web Experience Management Framework.* In this guide, the environment is also called *Web Experience Management Framework,* or *WEM Framework.*

The Site Capture application integrates with Oracle WebCenter Sites according to specifications in the *Oracle WebCenter Sites 11g Release 1 (11.1.1.x) Certification Matrix*. For additional information, see the release notes for WebCenter Sites. Check the WebCenter Sites documentation site regularly for updates to the *Certification Matrix* and release notes.

Audience

The Site Capture application and this guide are intended for general administrators and developers of WebCenter Sites. The Site Capture interface can be navigated by any of the above users, and its crawlers can be easily configured with basic code to download

websites. Writing advanced configuration code, as discussed in the concluding chapter, requires the expertise of a Java developer.

Using the Site Capture application as described in this guide requires you to have the following permissions and experience:

- You must be a WebCenter Sites user with the `GeneralAdmin` role and all the credentials of a general administrator who also belongs to the `RestAdmin` security group.
- You have administrative access to the Site Capture host machine. You will be using the Site Capture file system to access statically captured sites and related information.
- If you will be writing advanced configuration code, you must have expertise in Java.

Related Documents

For more information, see the following documents:

- *Oracle WebCenter Sites Installation Guide for the Site Capture Application*
- *Oracle WebCenter Sites Administrator's Guide*

Conventions

The following text conventions are used in this guide:

- **Boldface** type indicates graphical user interface elements that you select.
- *Italic* type indicates book titles, emphasis, or variables for which you supply particular values.
- `Monospace` type indicates file names, URLs, sample code, or text that appears on the screen.
- **`Monospace bold`** type indicates a command.

Steps in This Guide

Some of the steps in this guide are written as "quick steps" to provide you with a quick reference on completing various operations. For example, to view the crawler's list of archived sites, you would see the following quick steps:

Crawlers > *crawlerName* > Archives

The steps above mean:

Go to the "Crawlers" home page, point to a crawler, and select **Archives** from the pop-up menu.

When concepts, features, and the associated operations require explanation, the steps are written in detail.

Third-Party Libraries

Oracle WebCenter Sites and its applications include third-party libraries. For additional information, see *Oracle WebCenter Sites 11gR1: Third-Party Licenses*.

Chapter 1

Welcome to Oracle WebCenter Sites: Site Capture

This chapter introduces the Oracle WebCenter Sites: Site Capture application and shows you how to navigate its interface.

This chapter contains the following topics:

Site Capture Model

Crawls can be initiated manually from the Site Capture interface, or they can be triggered by the completion of a WebCenter Sites RealTime publishing session. In each scenario, the crawler downloads the website to disk in one of the following modes: static or archive, depending on how you choose to run the crawler.

Capture Modes

When a site is downloaded in either static or archive mode, the same files (html, css, and so on) are stored to disk, but with several differences. For example, statically downloaded sites are available only in the file system, whereas archived sites are available in both the file system and the Site Capture interface. Capture mode, then, determines how crawlers download sites and how you manage the results.

Static Mode	Archive Mode
Static mode supports rapid deployment, high availability scenarios.	Archive mode is used to maintain copies of websites on a regular basis for compliance purposes or similar reasons.
In static mode, a crawled site is stored as files ready to be served. Only the latest capture is kept (the previously stored files are overwritten).	In archive mode, all crawled sites are kept and stored as zip files (archives) in time-stamped folders. Pointers to the zip files are created in the Site Capture database.
Static crawl sessions can be initiated manually from the application interface or at the end of a publishing session. However, the downloaded sites can be accessed and managed only from the Site Capture file system.	Archive crawl sessions, like static sessions, can be initiated manually from the Site Capture interface or at the end of a publishing session. However, because the zip files are referenced by pointers in the Site Capture database, they can be managed from the Site Capture interface. Here, you can download the files, preview the archived sites, and set capture schedules.

For any capture mode, logs are generated at the end of the crawl session to provide such information as crawled URLs, HTTP status, and network conditions. In static capture, the logs must be obtained from the file system. In archive capture, they can be downloaded from the Site Capture interface. For any capture mode, you have the option of configuring crawlers to email reports as soon as they are generated.

Crawlers

Starting any type of site capture process requires a crawler to be defined in the Site Capture interface. To help you get started quickly, Site Capture comes with two sample crawlers, “Sample” and “FirstSiteII.” We assume the crawlers were installed during the Site Capture installation process. This guide uses mainly the “Sample” crawler.

Creating your own crawler involves naming the crawler (typically, after the target site), and uploading a text file named `CrawlerConfigurator.groovy`, which controls the crawler's site capture process. The `groovy` file must be coded with methods in the `BaseConfigurator` class that specify at least the starting URI(s) and link extraction logic for the crawler. Although the `groovy` file controls the crawler's site capture process, the crawler's capture mode is set outside the file.

Using a crawler for publishing-triggered site capture requires taking an additional step – you will name the crawler and specify its capture mode on the publishing destination definition on the WebCenter Sites source system that is integrated with Site Capture, as described in the *Oracle WebCenter Sites Installation Guide for the Site Capture Application*. (Bear in mind that on every publishing destination definition, you can specify one or more crawlers, but only a single capture mode.) Information about the success of crawler invocation is stored in the Site Capture file system and in the log files (`futuretense.txt`, by default) of the WebCenter Sites source and target systems.

The exercises in this chapter cover both types of crawler invocation scenarios: manual and publishing-triggered.

Logging in to the Site Capture Application

The Site Capture application runs on WebCenter Sites. You will access the Site Capture application by logging in to WebCenter Sites.

To log in to the Site Capture application

1. Access WebCenter Sites at the following URL:

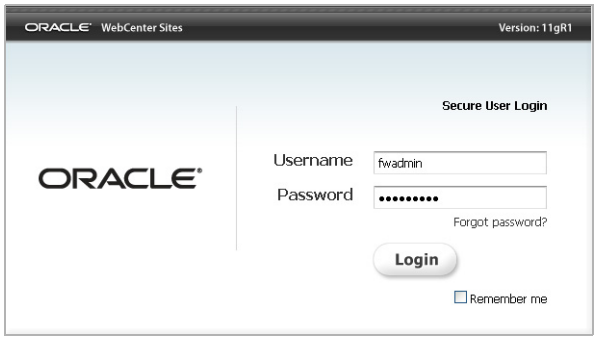
   `http://<server>:<port>/<context>/login`

   where `<server>` is the host name or IP address of the server running WebCenter Sites, `<port>` is the number of the WebCenter Sites application, and `<context>` is the name of the WebCenter Sites web application that was deployed on the server.

2. Log in as a general administrator. **Login credentials are case sensitive.** In this guide, we use the default credentials:

   Username: `fwadmin`

   Password: `xceladmin`

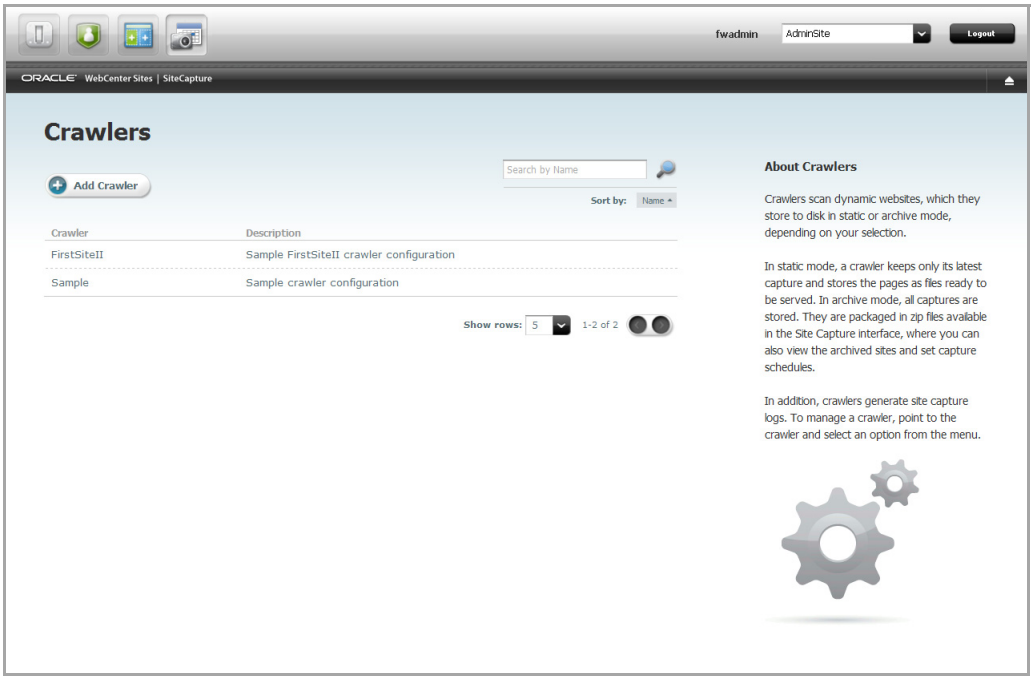


3. Click **Login**.

4. If you are logging in for the first time, the following screen opens:

Select the AdminSite (to which the Site Capture application is assigned by default) and select the Site Capture icon.

5. The first screen you see is named “Crawlers.” If the default crawlers were installed with Site Capture, you will see them listed under the names **Sample** and **FirstSiteII**.



6. Your next step can be any one of the following, depending on your requirements:
   - If you wish to learn more about the default crawlers, continue to "Using the Default Crawlers," on page 12.
   - To set up your own site capture operation and, in the process, learn to navigate the Site Capture interface, skip to "Setting Up a Site Capture Operation," on page 13.
   - To learn about crawler configuration code, see Chapter 3, "Coding the Crawler Configuration File."

Using the Default Crawlers

In this guide, we assume that the default crawlers "Sample" and "FirstSiteII" were installed with the Site Capture application, and they are displayed in its interface (as shown in step 5 on page 10). If you wish to define your own crawlers, see "Setting Up a Site Capture Operation," on page 13.

Sample Crawler

The "Sample" crawler can be used to download any site. The purpose of the "Sample" crawler is to help you quickly download the site and to provide you with required configuration code, which you will reuse when creating your own crawlers. The "Sample" crawler is minimally configured with the required methods and an optional method that limits the duration of the crawl by limiting the number of links to crawl.

- The required methods are `getStartURi` and `createLinkExtractor` (which defines the logic for extracting links from crawled pages).
- The optional method is `getMaxLinks`, which specifies the number of links to crawl.

More information about these methods as well as crawler customization methods and interfaces is available in Chapter 3, "Coding the Crawler Configuration File."

FirstSiteII Crawler

The "FirstSiteII" crawler is used to download WebCenter Sites' dynamic FirstSiteII sample website as a static site. The purpose of the crawler is to provide you with advanced configuration code that shows how to create a custom link extractor and resource rewriter, using the `LinkExtractor` and `ResourceRewriter` interfaces. More information about the interfaces is available in Chapter 3, "Coding the Crawler Configuration File."

Running a Default Crawler

In this section, you will run either the Sample crawler or the FirstSiteII crawler. Using the "FirstSiteII" crawler requires WebCenter Sites' FirstSiteII sample site to be published.

To run a default crawler

1. On the "Crawlers" screen, point to one of the default crawlers – **Sample** or **FirstSiteII** – and select **Edit Configuration**.

   Note

   If the default crawlers are not listed, skip to "Setting Up a Site Capture Operation," on page 13 to define your own crawler.

2. Set the crawler's starting URI by editing the crawler's configuration file. For instructions, skip to step 2 on page 16, and continue with the rest of the steps to run the crawler and manage its captured data.

Setting Up a Site Capture Operation

In this section, you will step through the process of creating and running your own crawler to understand how the Site Capture interface and file system are organized. Your basic steps are the following:

Step 1: Creating a Starter Crawler Configuration File

Before you can create a crawler, you must have a configuration file that controls the crawler's site capture process. The fastest way to create a useful file is to copy sample code and recode, as necessary.

To create a starter crawler configuration file

1. Copy the **Sample** crawler's configuration file to your local machine in one of the following ways:
   - Log in to the Site Capture application. If the "Crawlers" screen lists the "Sample" crawler, do the following (otherwise, skip to the item directly below):
     a) Point to **Sample** and select **Edit Configuration**.
     b) Go to the "Configuration File" field, copy its code to a text file on your local machine, and save the file as `CrawlerConfigurator.groovy`.
   - Go to the Site Capture host machine and copy the `CrawlerConfigurator.groovy` file from `<SC_INSTALL_DIR>/fw-site-capture/crawler/Sample/app/` to your local machine.

Note

Every crawler is controlled by its own `CrawlerConfigurator.groovy` file. The file is stored in a custom folder structure. For example:

When you define a crawler, Site Capture creates a folder bearing the name of the crawler (`<crawlerName>`, or `Sample` in our scenario) and places that folder in the following path: `<SC_INSTALL_DIR>/fw-site-capture/crawler/`. Within the `<crawlerName>` folder, Site Capture creates an `/app` subfolder to which it uploads the `groovy` file from your local machine.

When the crawler is used for the first time in a given mode, Site Capture creates additional subfolders (in `/<crawlerName>/`) to store sites captured in that mode. For more information about the Site Capture file system, see "Managing Statically Captured Sites," on page 30.

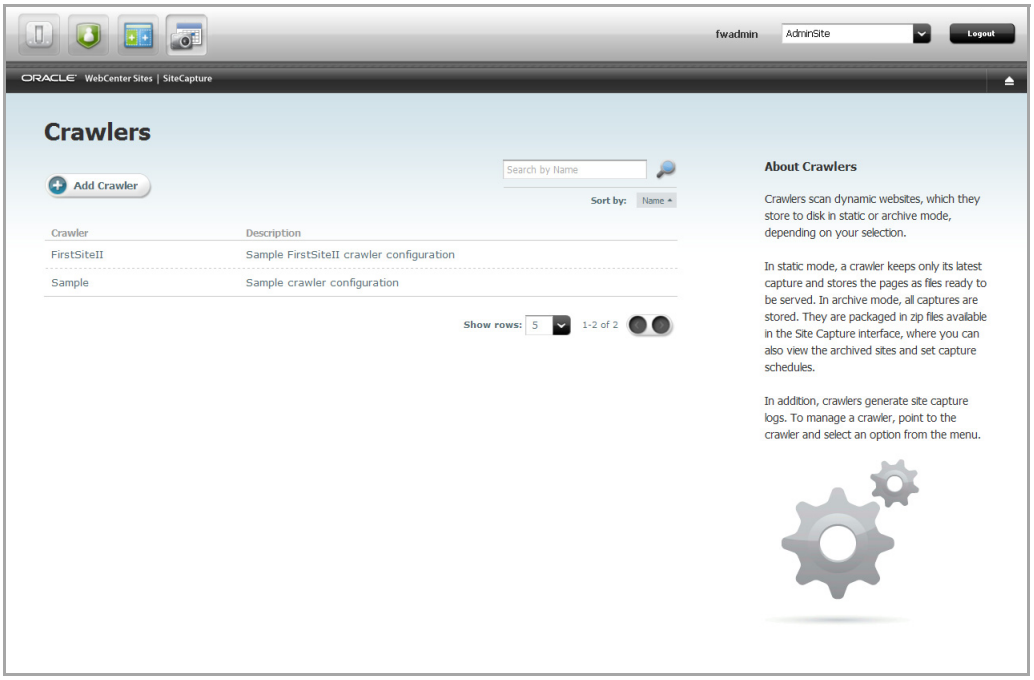
2. Your sample `groovy` file specifies a sample starting URI, which you will reset for the crawler you will be creating in the next step. (In addition to the starting URI, you can set crawl depth and similar parameters, invoke post-crawl commands, and implement interfaces to define logic specific to your target sites.)

   At this point, you have the option to either customize the downloaded `groovy` file now, or first create the crawler and then customize its `groovy` file (which is editable in the Site Capture interface).

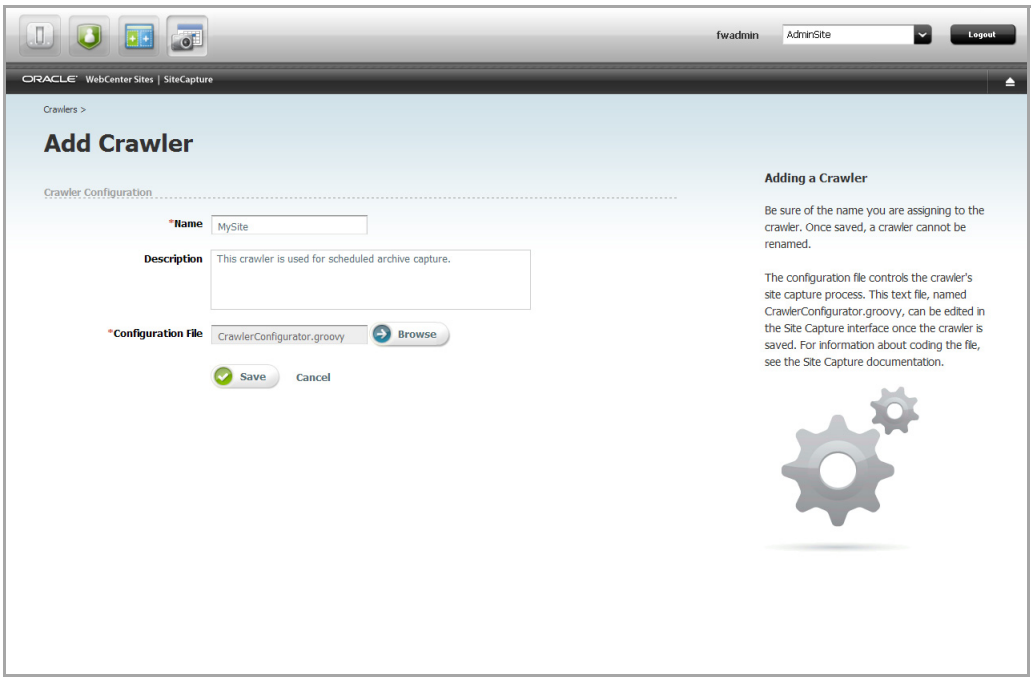
   - To follow this exercise, continue to the next step "Step 2: Defining a Crawler."
   - For information about crawler configuration methods and interfaces, see Chapter 3, "Coding the Crawler Configuration File."

Step 2: Defining a Crawler

1. Go to the "Crawlers" screen and click **Add Crawler**.



2. On the "Add Crawlers" screen:



   a. Name the crawler after the site to be crawled.

   Note

   - Once saved, a crawler cannot be renamed.
   - Throughout this guide, we assume that every custom crawler is named after the target site and will not be used to capture any other site.

   b. Enter a description (optional). For example: "This crawler is reserved for publishing-triggered site capture" or "This crawler is reserved for scheduled captures."

   c. In the "Configuration File" field, browse to the `groovy` file that you created in "Step 1: Creating a Starter Crawler Configuration File," on page 13.

   d. Save the new crawler.

      Your `CrawlerConfigurator.groovy` file is uploaded to the `<SC_INSTALL_DIR>/fw-site-capture/crawler/<crawlerName>/app` folder on the Site Capture host machine. The file can be edited directly in the Site Capture interface.

3. Continue to "Step 3: Editing the Crawler Configuration File," on page 16.

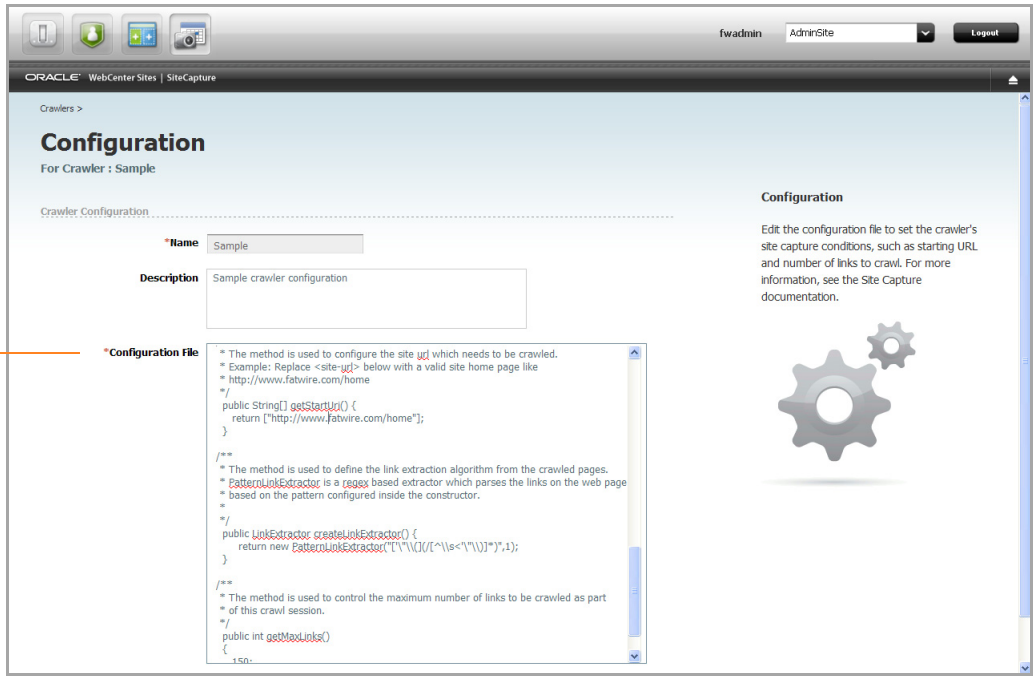
Step 3: Editing the Crawler Configuration File

From the Site Capture interface, you can recode the entire crawler configuration file. In this example, we simply set the crawler's starting URI.

To edit the crawler configuration file

1. On the "Crawlers" screen, point to the crawler you just defined and select **Edit Configuration**.

This field displays the crawler's `CrawlerConfigurator.groovy` file, located in `<SC_INSTALL_DIR>/fw-site-capture/crawler/<crawlerName>/app`.



2. Set the crawler's starting URI in the following method:

```
public String[] getStartUri() {
  return ["http://www.mycompany.com/home"]
```

Note

- You can set multiple starting URIs. They must belong to the same site. Enter a comma-separated array, as shown in the example below:

```
public String[] getStartUri()
 {
  return ["http://www.fatwire.com/product","http://
   www.fatwire.com/support"];
 }
```

- Your configuration file includes the `createLinkExtractor` method, which calls the logic for extracting the links to be crawled. The links will be extracted from the markup that is downloaded during the crawl session. For additional information about this method and the extraction logic, see "createLinkExtractor," on page 39.

Note *(continued)*

- Your configuration file also includes the `getMaxLinks()` method, which specifies the number of links to crawl. Its default value is set to `150` to ensure a quick run. If for some reason you have to stop a static capture, you will have to stop the application server. Archive captures can be stopped from the Site Capture interface.

For information about crawler configuration methods and interfaces, see Chapter 3, "Coding the Crawler Configuration File."

3. Click **Save**.
4. Continue to "Step 4: Starting a Crawl," on page 17.

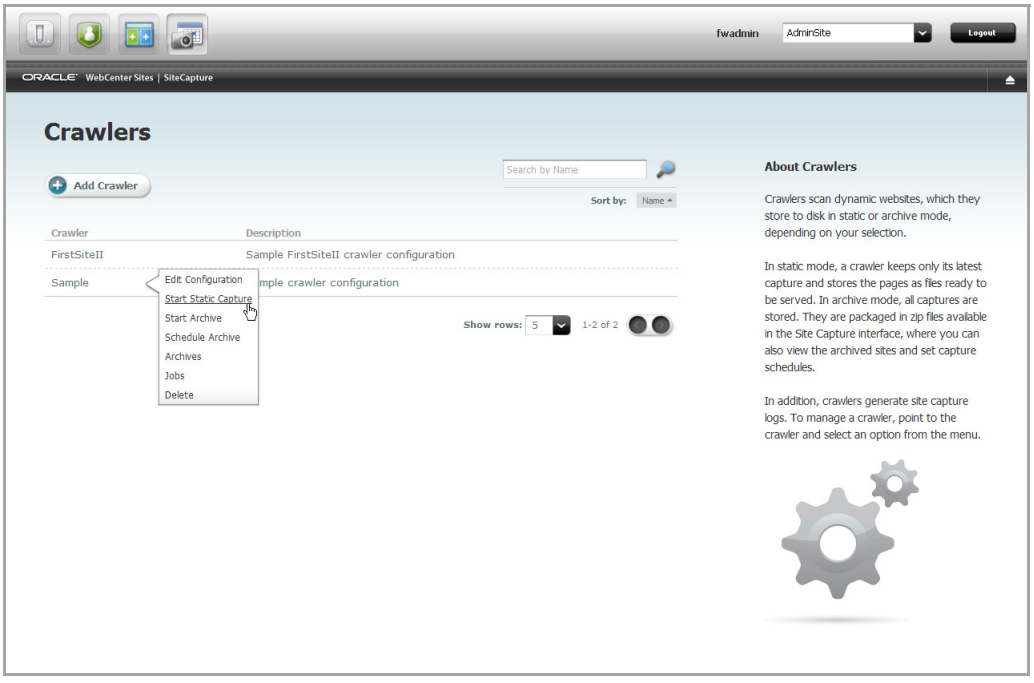
Step 4: Starting a Crawl

You can start a crawl in several ways:

- Run the Crawler Manually in Static Mode
- Run the Crawler Manually in Archive Mode
- Schedule the Crawler for Archive Capture (static capture cannot be scheduled)
- Publish a Site in RealTime Mode

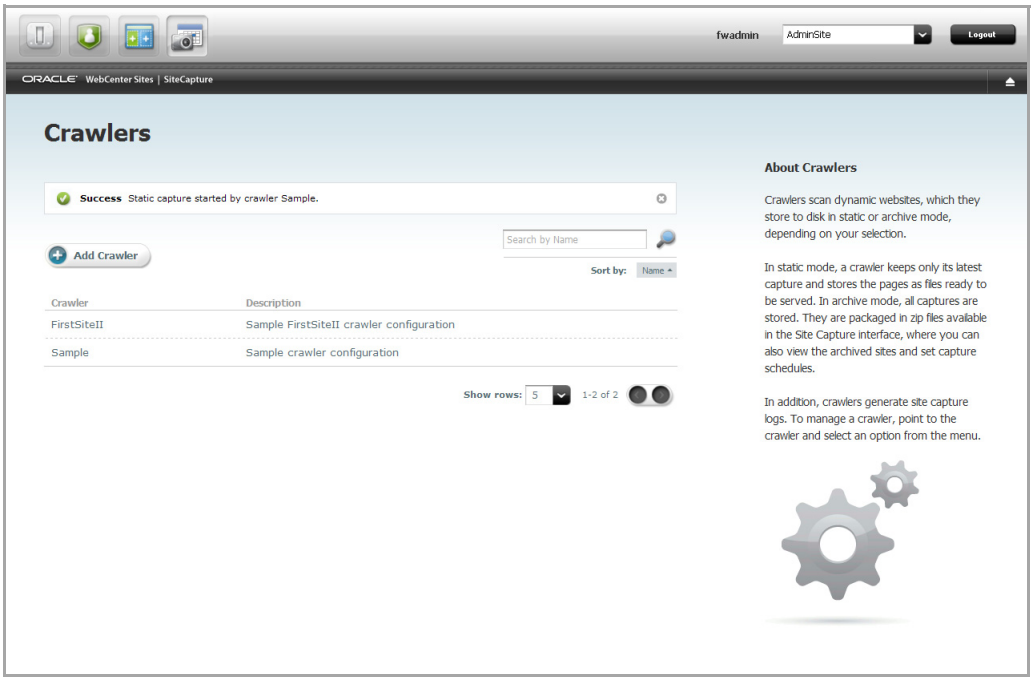
Run the Crawler Manually in Static Mode

1. On the "Crawlers" screen, point to the crawler that you created and select **Start Static Capture** from the pop-up menu.



The "Crawlers" screen displays the following message when capture begins:

"**Success.** `Static capture started by crawler <crawlerName>.`"



2. At this point, the Site Capture interface does not display any other information about the crawler or its process, nor will it make the downloaded site available to you. Instead, you will use the Site Capture file system to access the downloaded files and various logs:
   - To monitor the static capture process, look for the following files:
     - The `lock` file in `<SC_INSTALL_DIR>/fw-site-capture/<crawlerName>/logs`. The `lock` file is transient. It is created at the start of the static capture process to prevent the crawler from being invoked for an additional static capture. The `lock` file is deleted when the crawl session ends.
     - The `crawler.log` file in `<SC_INSTALL_DIR>/fw-site-capture/logs/`. (This file uses the term "VirtualHost" to mean "crawler.")
     - The `inventory.db` file in `<SC_INSTALL_DIR>/fw-site-capture/<crawlerName>`. This file lists the crawled URLs. **The `inventory.db` file is used by the Site Capture system and must not be deleted or modified.**
     - The `audit.log`, `links.txt` file, and `report.txt` files are available in `/fw-site-capture/crawler/<crawlerName>/logs/yyyy/mm/dd`.
   - To access the downloaded files, go to `<SC_INSTALL_DIR>/fw-site-capture/crawler/<crawlerName>/www`.

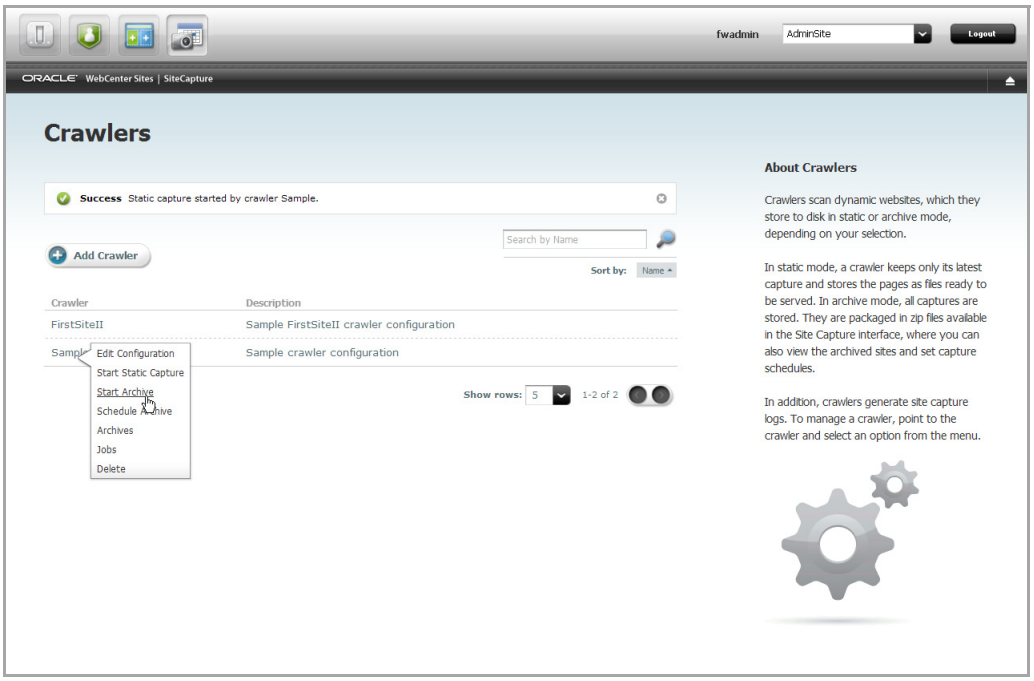
   For more information about the Site Capture file system, see "Managing Statically Captured Sites," on page 30.

Run the Crawler Manually in Archive Mode

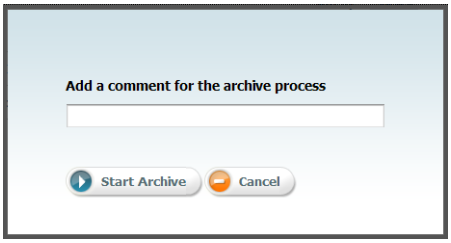
If a crawler was used in one mode, it can be rerun in a different mode.

To run the crawler in archive mode

1. On the "Crawlers" screen, point to the crawler that you created and select **Start Archive**.



2. In the next dialog:
   - You can add a comment about the crawler's upcoming job:



Note

A comment cannot be added once a crawler starts running.

If you choose to add a comment in the dialog above, it will be displayed in the following places:

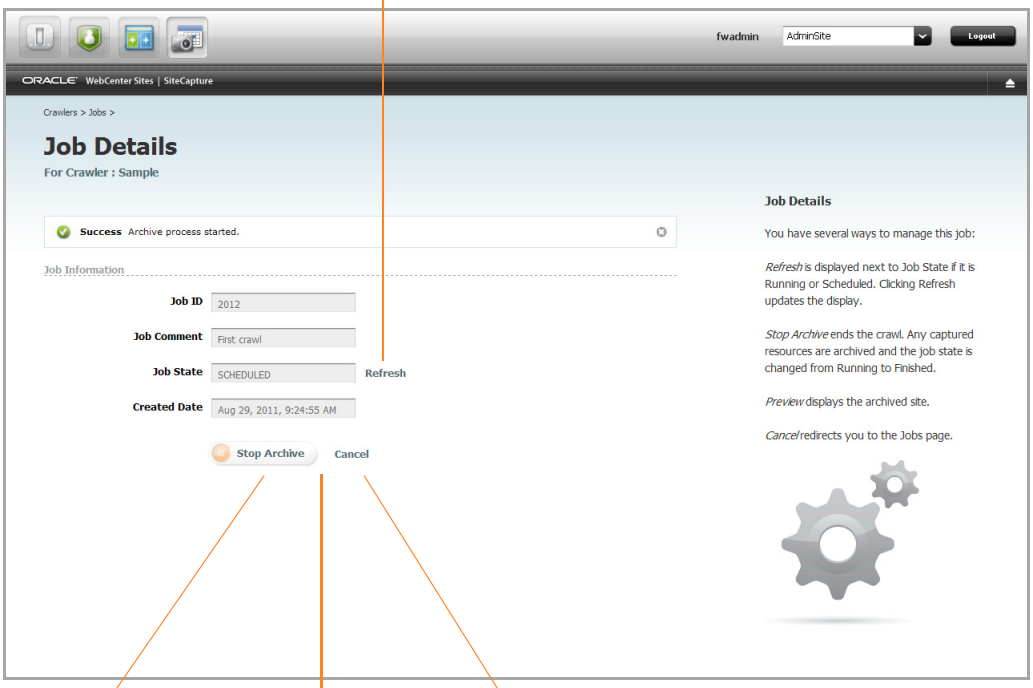
- "Job Details" screen, "Job Comment" field (shown in the next step).
- "Jobs" screen, "Job Comment" field (**Crawler > *crawlerName* > Jobs**).
- "Archives" screen, "Comment" field (**Crawler > *crawlerName* > Archives**).

- Click **Start Archive**.

3. You are now viewing the "Job Details" screen, where you can manage the archive process in several ways, as shown in the figure below. To follow this exercise, click **Refresh** (next to "Job State") until "Finished" is displayed, then continue to the next step.

The "Job Details" screen displays several options for managing archive capture:

Refresh is shown as long as the job state is Scheduled or Running. Clicking **Refresh** updates the displayed job state. Possible job states are Scheduled, Running, Finished, Stopped, and Failed.

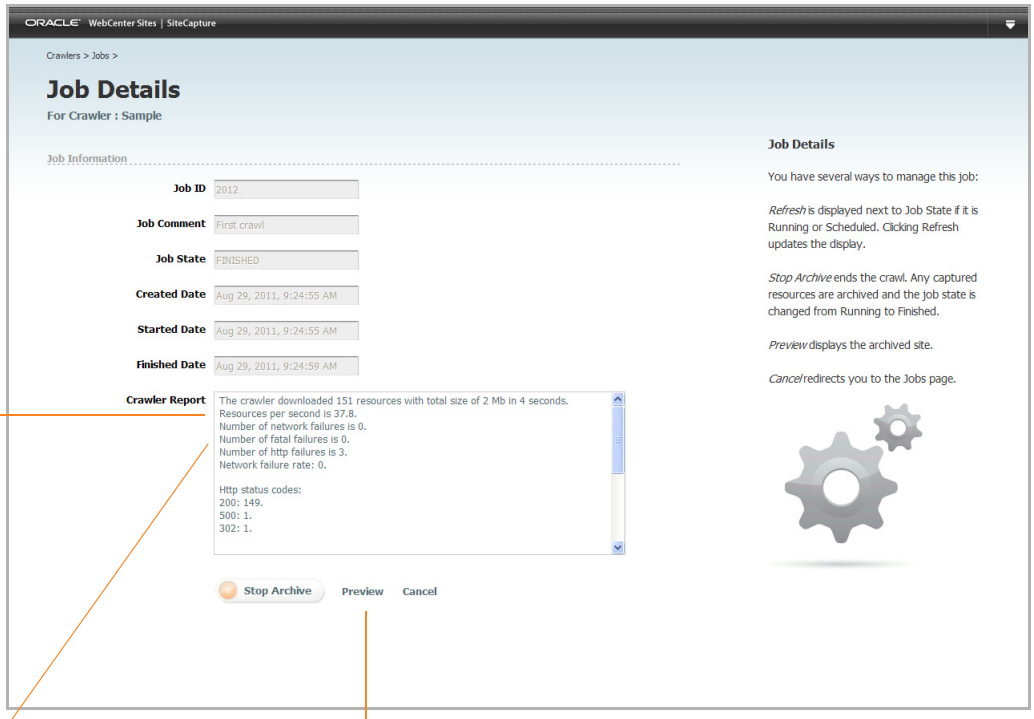


Stop Archive ends the crawler's session. Any captured resources are archived and the job state is changed from "Running" to "Finished" (click **Refresh** to see the change).

Cancel redirects you to the "Jobs" screen. The crawler, if running, continues to run.

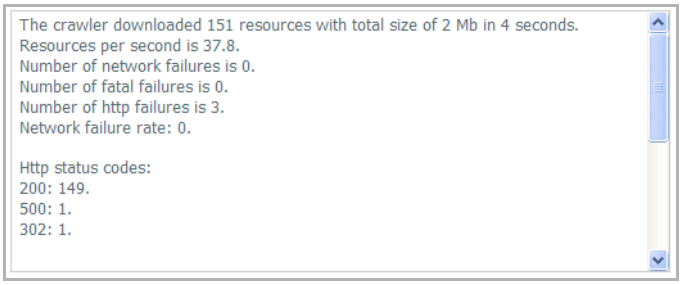
Preview will be displayed here when the job state is "Finished." Clicking **Preview** displays the archived site.

4. When the archive crawl ends, results are made available in the Site Capture interface. For example:
   - The crawler report is displayed on the "Job Details" screen. The report lists the number of downloaded resources, their total size and download time, network conditions, HTTP status codes, and additional notes as necessary.



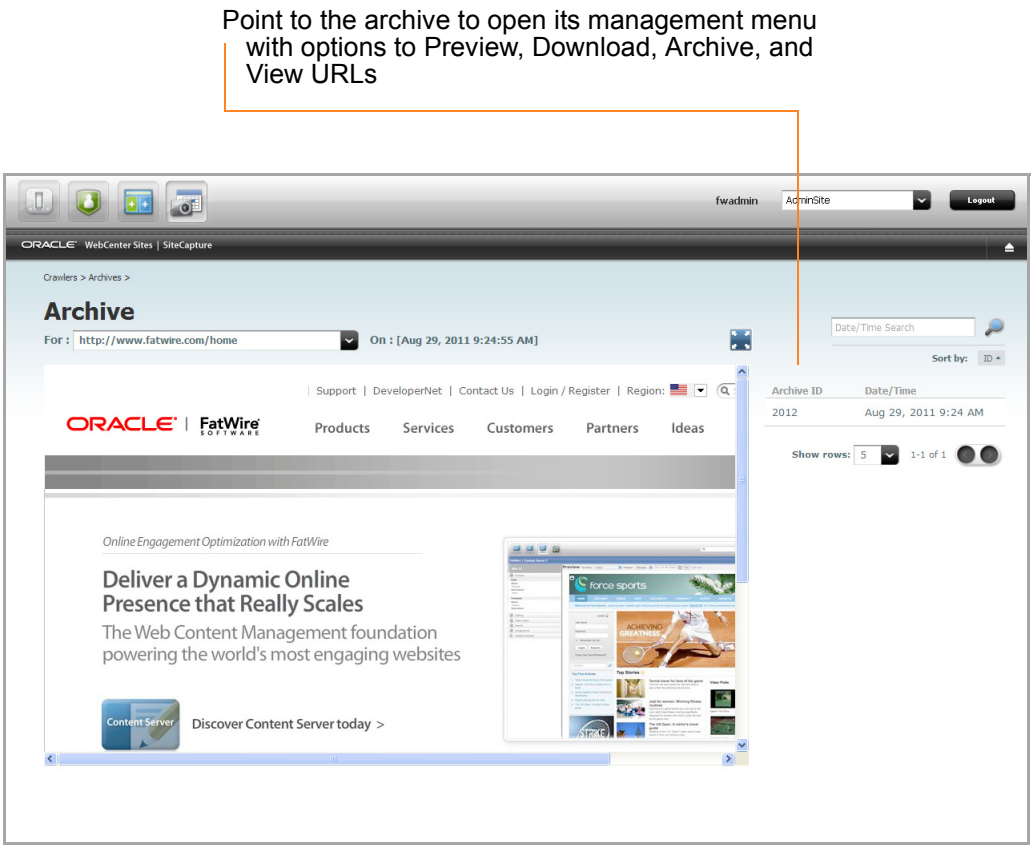
Crawler Report

Clicking **Preview** renders the archived site



- Clicking **Preview** on the "Job Details" screen renders the archived site (as shown in the figure below). Next to the site is the Archive ID table with archive management options, which are displayed when you point to an archive.

> **Note**
>
> If your archived site contains links to external domains, its preview is likely to include those links, especially when the crawl depth and number of links to crawl are set to large values (in the `CrawlerConfigurator.groovy` file). Although the external domains can be browsed, they are not archived.



- For a summary of pathways to various data, see "Managing Archived Sites," on page 33.

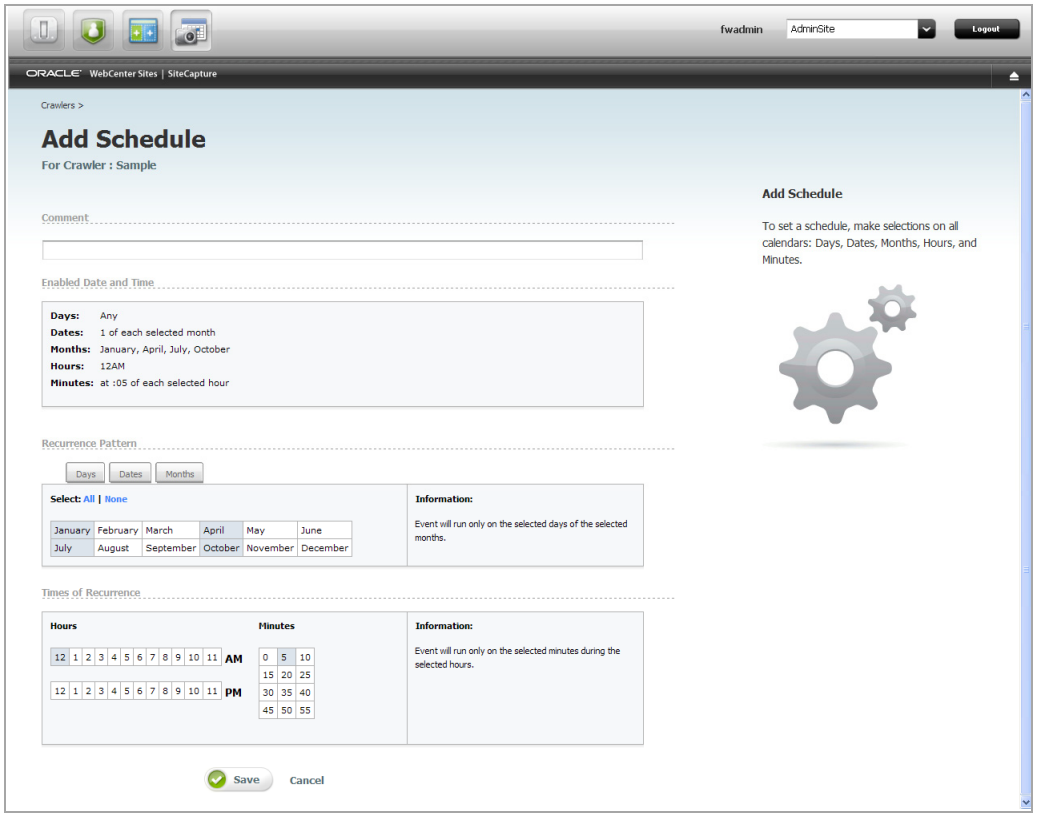
Schedule the Crawler for Archive Capture

Only archive captures can be scheduled. For a given crawler, you can create multiple schedules – for example, one for capturing periodically, and another for capturing at a particular *and* unique time.

> **Note**
>
> If you set multiple schedules, ensure they do not overlap.

To schedule a crawler for archive capture

1. Go the "Crawlers" screen, point to the crawler that you created and select **Schedule Archive**.
2. Click **Add Schedule** and make selections on all calendars: Days, Dates, Months, Hours, and Minutes.



3. Click **Save** and add another schedule if necessary.

Publish a Site in RealTime Mode

If you configure your WebCenter Sites publishing systems to communicate with the Site Capture application, you can set up a RealTime publishing process to invoke one or more crawlers to capture the newly published site. For instructions, see "Enabling Publishing-Triggered Site Capture," on page 25.

Step 5: Managing Captured Data

Information about accessing various data associated with static and archive captures can be found in Chapter 2, "Managing Downloaded Sites." The "Summary," on page 34 presents a collection of notes and tips to bear in mind when managing crawlers and captured data. Topics include the following:

- Creating and Editing Crawlers
- Deleting a Crawler
- Scheduling a Crawler
- Monitoring a Static Crawl
- Stopping a Crawl
- Downloading Archives
- Previewing Sites
- Configuring Publishing Destination Definitions
- Accessing Log Files

Enabling Publishing-Triggered Site Capture

Your main steps for enabling publishing-triggered site capture are the following:

1. Integrating the Site Capture Application with Oracle WebCenter Sites
2. Configuring a RealTime Publishing Destination Definition for Site Capture
3. Matching Crawlers

An administrative user can configure as many publishing destination definitions for Site Capture as necessary and invoke as many crawlers as necessary.

Integrating the Site Capture Application with Oracle WebCenter Sites

You can enable site capture at the end of a RealTime publishing session only if the Site Capture application is first integrated with the WebCenter Sites source and target systems used in the publishing process. If Site Capture is not integrated, see the *Oracle WebCenter Sites Installation Guide for the Site Capture Application* for integration instructions, then continue with the steps below.

Configuring a RealTime Publishing Destination Definition for Site Capture

When configuring a publishing destination definition, you will name the crawler(s) that will be invoked at the end of the publishing session. You will also specify the capture mode.

To configure the publishing destination definition

1. Go to the WebCenter Sites source system that is integrated with the Site Capture application (as shown in the *WebCenter Sites Installation Guide for the Site Capture Application*).
   a. Create a RealTime publishing destination definition pointing to the WebCenter Sites target system that is integrated with Site Capture. (For instructions on creating a RealTime publishing destination, see the *Oracle WebCenter Sites Administrator's Guide*.)
   b. In the "More Arguments" section of the publishing destination definition, name the crawler(s) to be invoked at the end of the publishing session, and set the capture mode by using the following parameters to control crawler invocation:
      - `CRAWLERCONFIG`: Specify the name of each crawler. If you are using multiple crawlers, separate their names with a semicolon (;).

        Examples:
        For a single crawler: `CRAWLERCONFIG=crawler1`
        For multiple crawlers: `CRAWLERCONFIG=crawler1;crawler2;crawler3`

> **Note**
>
> The crawler(s) that you specify here must also be configured and identically named in the Site Capture interface. Crawler names are case sensitive.

- `CRAWLERMODE`: To run an archive capture, set this parameter to `dynamic`. By default, static capture is enabled.

  Example: `CRAWLERMODE=dynamic`

Note

- If `CRAWLERMODE` mode is omitted or set to a value other than `dynamic`, static capture will start when the publishing session ends.
- Both crawler parameters can be set in a single statement as follows: `CRAWLERCONFIG=crawler1;crawler2&CRAWLERMODE=dynamic`
- While you can specify multiple crawlers, you can set only one mode. All crawlers will run in that mode. To run some of the crawlers in a different mode, configure another publishing destination definition.

2. Continue to the next section.

Matching Crawlers

Crawlers named in the publishing destination definition must exist in the Site Capture interface. Do the following:

- Verify that crawler names in the destination definition (step 1b on page 25) and Site Capture interface are identical. The names are case sensitive.
- Ensure that a valid starting URI for the target site is set in each crawler's configuration file. For information about navigating to the crawler's configuration file, see "Step 3: Editing the Crawler Configuration File," on page 16. For more information about writing configuration code, see Chapter 3, "Coding the Crawler Configuration File."

Next Steps

1. Having enabled publishing-triggered site capture, you are ready to publish the target site. When publishing ends, site capture begins. The invoked crawlers capture pages in either static or archive mode, depending on how you set the `CRAWLERMODE` parameter in the publishing destination definition (step 1b on page 25).
2. To monitor the site capture process.
   - For static capture, the Site Capture interface does not display any information about the crawl, nor will it make the captured site available to you.
     - To determine that the crawlers were invoked, open the `futuretense.txt` file on the source or target WebCenter Sites system.

       Note

       The `futuretense.txt` file on the WebCenter Sites source and target systems contains crawler invocation status for any type of crawl: static and archive.

     - To monitor the capture process, go to the Site Capture file system and review the files that are listed on page 18.

- For dynamic capture, you can view the status of the crawl from the Site Capture interface.

  a) Go to the "Crawlers" screen, point to the crawler, and select **Jobs** from the pop-up menu.

  b) On the "Job Details" screen, click **Refresh** next to the "Job State" until you see "Finished." (Possible value for "Job State" are Scheduled, Running, Finished, Stopped, or Failed.) For more information about the "Job Details" screen, see pages 20 and 21.

3. Managing captured data.

When the crawl session ends, you can manage the captured site and associated data as follows:

- For a statically captured site, go to the Site Capture file system. For more information, see "Managing Statically Captured Sites," on page 30."
- For an archived site, use the Site Capture interface to preview the site and download the zip file and logs. For more information, see "Managing Archived Sites," on page 33.

Chapter 2

Managing Downloaded Sites

Downloaded sites are managed either from the Site Capture file system or the interface, depending on whether they are statically captured or archived.

This chapter contains the following topics:

Managing Statically Captured Sites

For every crawler that a user creates in the Site Capture interface, Site Capture creates an identically named folder in its file system. This custom folder, `<crawlerName>`, is used to organize the crawler's configuration file, captures, and logs as shown in Figure 1. Table 1 describes the `<crawlerName>` folder and its contents.

Note

To access static captures and logs, you will have to use the file system. Archive captures and logs are managed from the Site Capture interface (their location in the file system is included in this section).

Figure 1: Site Capture's Custom Folders: `<crawlerName>`

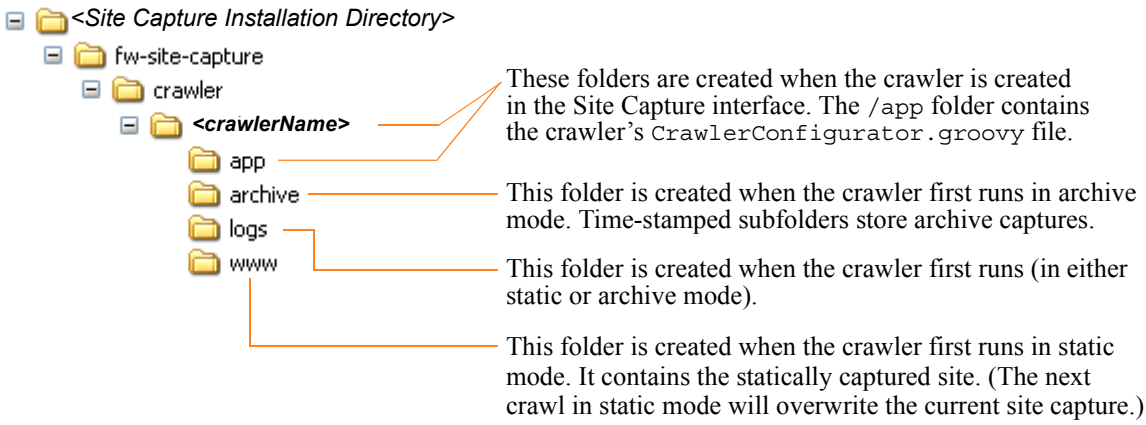


Table 1: A `<crawlerName>` Folder and Its Contents

Folder	Description
`/fw-site-capture/crawler/<crawlerName>`	Represents a crawler. For every crawler that a user defines in the Site Capture interface, Site Capture creates a `/<crawlerName>` folder. For example, if you installed the sample crawlers "FirstSiteII" and "Sample," you will see both crawlers listed in the Site Capture interface, and you will find identically named folders in the Site Capture file system. **Note:** In addition to the subfolders (described below), the `<crawlerName>` folder contains an `inventory.db` file, which lists statically crawled URLs. The file is created when the crawler takes its first static capture. **Do not delete or modify `inventory.db`**. It is used by the Site Capture system.
`/fw-site-capture/crawler/<crawlerName>/app`	Contains the crawler's `CrawlerConfiguration.groovy` file. Its code controls the crawl process. The `/app` folder is created when the crawler is created and saved.

Table 1: A `<crawlerName>` Folder and Its Contents *(continued)*

Folder	Description
`/fw-site-capture/crawler/<crawlerName>/archive`	The `/archive` folder is used strictly for archive capture. This folder contains a hierarchy of `yyyy/mm/dd` subfolders. The `/dd` subfolder stores all of the crawler's archive captures in time-stamped zip files. The `/archive` folder is created when the crawler first runs in archive mode. The zip files (located in `/dd`) are referenced in the database and therefore made available in the Site Capture interface for you to download and display as websites. **Note:** Archive captures are accessible from the Site Capture interface. Each zip file contains a URL log named `__inventory.db`. **Do not delete or modify `__inventory.db`**. It is used by the Site Capture system.
`/fw-site-capture/crawler/<crawlerName>/www`	Contains only the latest *statically* captured site (when the same crawler is rerun in static mode, it overwrites the previous capture). The site is stored as `html`, `css`, and other files that can be readily served. The `/www` folder is created when the crawler first runs in static mode. **Note:** Static captures are accessible only from the Site Capture file system.

`/fw-site-capture/crawler/<crawlerName>/logs/yyyy/mm/dd`	Contains log files with information about crawled URLs. Log files are stored in the `/dd` subfolders and named as shown in the figure below.
	 
	These folders contain the following logs:
	`<yyyy-mm-dd-hh-mm-ss>-audit.log`
	`<yyyy-mm-dd-hh-mm-ss>-links.txt`
	`<yyyy-mm-dd-hh-mm-ss>-report.txt`
	• The `audit.log` file lists the crawled URLs with data such as timestamps, crawl depth, HTTP status, and download time.
	• The `links.txt` file lists the crawled URLs.
	• The `report.txt` file lists the overall crawl statistics such as number of downloaded resources, total size, download size and time, and network conditions. For archive capture, this report is available in the Site Capture interface as the crawler report (on the "Job Details" screen. Paths to the "Job Details" screen are shown in Figure 2, on page 33).

Table 1: A `<crawlerName>` Folder and Its Contents *(continued)*

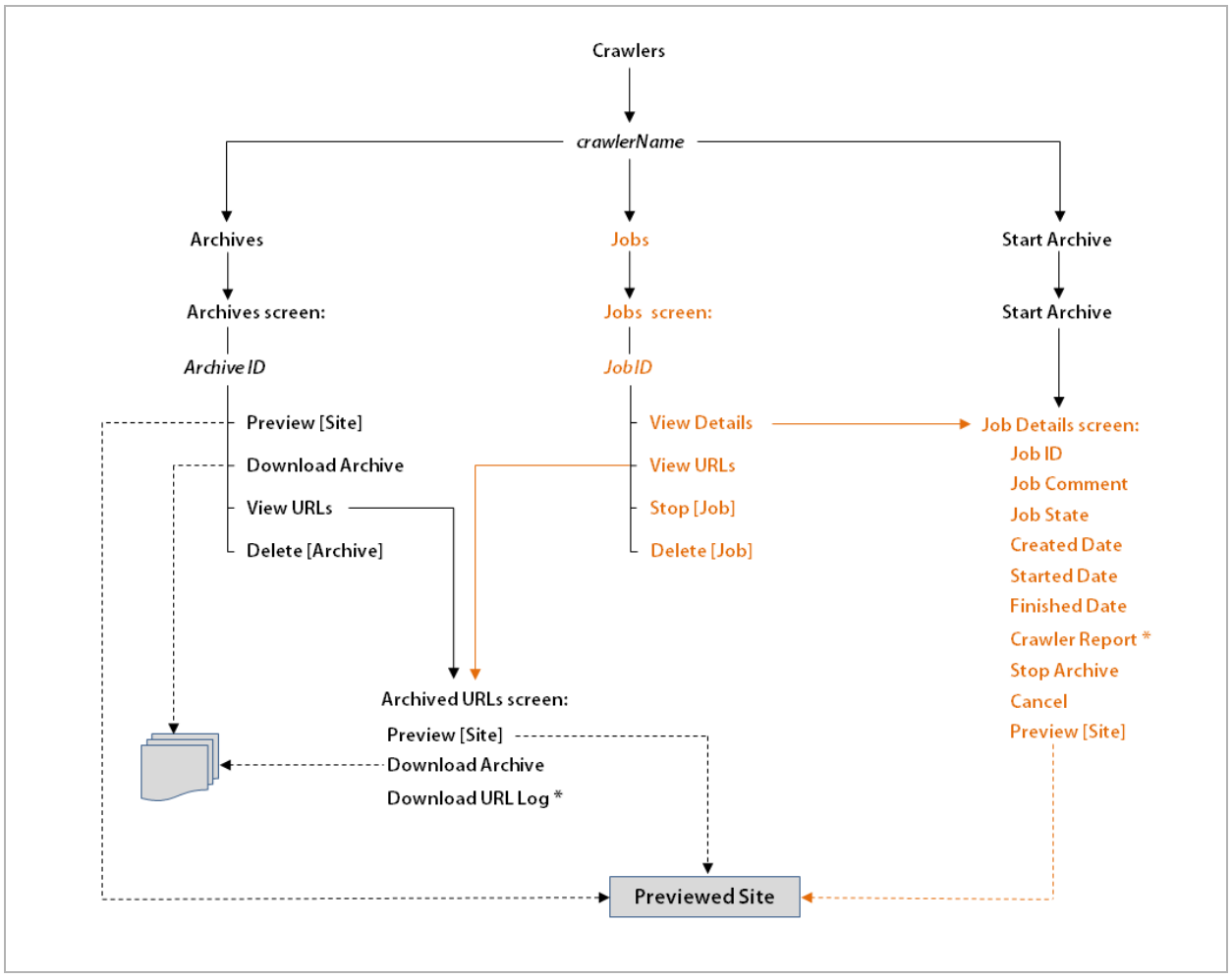
Folder	Description
`/fw-site-capture/crawler/<crawlerName>/logs` *(continued)*	**Note:** If the crawler captured in both static mode and archive mode, the `/dd` subfolders contain logs for static captures and archive captures. The `/logs` folder is also used to store a transient file named `lock`. The file is created at the start of the static capture process to prevent the crawler from being invoked for additional static captures. The `lock` file is deleted when the crawl session ends.

Managing Archived Sites

Archived sites can be managed from different screens of the Site Capture interface. Figure 2 shows some of the pathways to various information: archives, jobs, site preview, crawler report, and URL log:

- For example, to preview a site, start at the "Crawlers" screen, point to a crawler (***crawlerName***), select **Archives** from the pop-up menu (which opens the "Archives" screen), point to an **Archive ID**, and select **Preview** from the pop-up menu.
- Dashed lines represent multiple paths to the same option. For example, to preview a site, you can follow the crawler's **Archives** path, **Jobs** path, or **Start Archive** path. To download an archive, you can follow the **Archives** path or the **Jobs** path.
- The crawler report and URL log are marked by an asterisk (*).

Figure 2: Paths to Archived Information



Summary

This section summarizes notes and tips for managing crawlers and captured data.

Creating and Editing Crawlers

When creating crawlers and editing their configuration code, consider the following information:

- Crawler names are case sensitive.
- Every crawler's configuration file is named `CrawlerConfigurator.groovy`. The file is used to inject dependency. Hence, its name must not be changed.
- A crawler can be configured to start at one or more seed URIs on a given site and to crawl one or more paths. Additional Java methods enable you to set parameters such as crawl depth, invoke post-crawl commands, specify session timeout, and more. Interfaces can be implemented to define logic for extracting links, rewriting URLs, and sending email at the end of a crawl session. For more information, see Chapter 3, "Coding the Crawler Configuration File."
- When a crawler is created and saved, its `CrawlerConfigurator.groovy` file is uploaded to the Site Capture file system and made editable in the Site Capture interface.
- While a crawler is running a static site capture process, it cannot be invoked to run a second static capture process.
- While a crawler is running an archive capture process, it can be invoked to run a second archive capture process. The second process will be marked as "Scheduled" and will start after the first process ends.

Deleting a Crawler

If you need to delete a crawler (which includes all of its captured information), do so from the Site Capture interface, not the file system. Deleting from the interface prevents broken links. For example, if a crawler ran in archive mode, deleting it from the interface removes two sets of information: (1) the crawler's archives and logs, and (2) database references to those archives and logs. (Deleting the crawler from the file system retains database references to archives and logs that no longer exist, thus creating broken links in the Site Capture interface.)

Scheduling a Crawler

Only archive crawls can be scheduled.

- When setting a crawler's schedule, consider the site's publishing schedule and avoid overlapping the two.
- You can create multiple schedules for a single crawler – for example, one schedule to invoke the crawler periodically, and another schedule to invoke the crawler at one particular and unique time.
- When creating multiple schedules, ensure they do not overlap.

Monitoring a Static Crawl

To determine whether a static crawl is in progress or completed, look for the crawler's `lock` file in the `<SC_INSTALL_DIR>/fw-site-capture/<crawlerName>/logs` folder. The `lock` file is transient. It is created at the start of the static capture process to prevent the crawler from being invoked for an additional static capture. The `lock` file is deleted when the crawl session ends.

Stopping a Crawl

Before running a crawler, consider the number of links to be crawled and the crawl depth, both of which determine the duration of the crawler's session.

- If you need to terminate an archive crawl, use the Site Capture interface (**Job Details** screen > **Stop Archive**).
- If you need to terminate a static crawl, you will have to stop the application server.

Downloading Archives

Avoid downloading large archive files (exceeding 250MB) from the Site Capture interface. Instead, use the `getPostExecutionCommand` to copy the files from the Site Capture file system to your preferred location.

Archive size can be obtained from the crawler report, on the "Job Details" screen. Paths to the "Job Details" screen are shown in Figure 2, on page 33. For information about the `getPostExecutionCommand` method, see page 43.

Previewing Sites

If your archived site contains links to external domains, its preview is likely to include those links, especially when the crawl depth and number of links to crawl are set to large values (in the crawler's `groovy` file). Although the external domains can be browsed, they are not archived.

Configuring Publishing Destination Definitions

- If you are running publishing-triggered site capture, you can set crawler parameters in a single statement on the publishing destination definition:

  `CRAWLERCONFIG=`*`crawler1;crawler2`*`&CRAWLERMODE=dynamic`
- While you can specify multiple crawlers on a publishing destination definition, you can set only one capture mode. All crawlers will run in that mode. To run some of the crawlers in a different mode, configure another publishing destination definition.

Accessing Log Files

- For statically captured sites, log files are available only in the Site Capture file system:
  - The `inventory.db` file, which lists statically crawled URLs, is located in the `/fw-site-capture/crawler/<crawlerName>` folder.

Note

The `inventory.db` file is used by the Site Capture system. It must not be deleted or modified.

- The `crawler.log` file is located in the `<SC_INSTALL_DIR>/fw-site-capture/logs/` folder. (The `crawler.log` file uses the term "VirtualHost" to mean "crawler.")

- For statically captured and archived sites, a common set of log files exists in the site Capture file system:
  - `audit.log`,which lists the crawled URLs, timestamps, crawl depth, HTTP status, and download time.
  - `links.txt`, which lists the crawled URLs
  - `report.txt`, which is the crawler report

  The files named above are located in the following folder:
  `/fw-site-capture/crawler/<crawlerName>/logs/yyyy/mm/dd`

Note

For archived sites, `report.txt` is also available in the Site Capture interface, on the "Job Details" screen, where it is called the "Crawler Report." (Paths to the "Job Details" screen are shown in Figure 2, on page 33.

- The archive process also generates a URL log for every crawl. The log is available in two places:
  - In the Site Capture file system, where it is called `__inventory.db`. This file is located within the zip file in the following folder:
    `/fw-site-capture/crawler/<crawlerName>/archive/yyyy/mm/dd`

Note

The `__inventory.db` file is used by the Site Capture system. It must not be deleted or modified.

  - In the Site Capture interface, in the "Archived URLs" screen (whose path is shown in Figure 2, on page 33).

Chapter 3

Coding the Crawler Configuration File

This chapter contains information about the `BaseConfigurator` class, about implementing its methods and interfaces to control a crawler's site capture process, and about sample code that is available in the Site Capture installation for the FirstSiteII crawler.

This chapter contains the following topics:

Overview

Controlling a crawler requires coding its `CrawlerConfigurator.groovy` file with at least the following information: the starting URI and link extraction logic. Both pieces of information are supplied through the `getStartUri()` and `createLinkExtractor()` methods. Additional code can be added as necessary to specify, for example, the number of links to be crawled, the crawl depth, and the invocation of a post-crawl event such as copying statically downloaded files to a web server's doc base.

The methods and interfaces you will use are provided in the `BaseConfigurator` class. The default implementations can be overridden to customize and control a crawl process in a way that agrees with the structure of the target site and the data you need to collect.

This chapter begins with the `BaseConfigurator` methods and a simple `CrawlerConfigurator.groovy` file to demonstrate usage of the required methods. Crawler customization methods are then discussed and followed by information about Site Capture's Java interfaces, including their default and custom implementations.

BaseConfigurator Methods

The `CrawlerConfigurator.groovy` file contains the code of the `CrawlerConfigurator` class. This class must extend `BaseConfigurator`, which is an abstract class that provides default implementations for the crawler. Methods and interfaces of the `BaseConfigurator` class are listed in Table 1 and described in the sections that follow. A basic sample `CrawlerConfigurator.groovy` file is shown on page 40.

Table 1: Methods in the `BaseConfigurator` Class

Method Type	Method	Notes
Required	getStartUri	
	createLinkExtractor	Factory method in the LinkExtractor interface.[a,b]
Crawler Customization	getMaxLinks	
	getMaxCrawlDepth	
	getConnectionTimeout	
	getSocketTimeout	
	getPostExecutionCommand	
	getNumWorkers	
	getUserAgent	
	createResourceRewriter	Factory method in the ResourceRewriter interface.[a,b]
	createMailer	Factory method in the Mailer interface.[a]
	getProxyHost	
	getProxyCredentials	

a. The listed interfaces have default implementations, described in this chapter.

b. Site Capture provides a sample link extractor and resource rewriter, both used by the FirstSiteII sample crawler. See "Writing and Deploying a Custom Link Extractor" and "Writing a Custom ResourceRewriter."

Required Methods

Two abstract methods in `BaseConfigurator` must be overridden in `CrawlerConfigurator`. They are `getStartUri()` and `createLinkExtractor()`.

getStartUri

This method is used to inject the crawler's start URI. You can configure one or more start URIs for the crawl, as long as the URIs belong to the same site. If you specify multiple starting points, the crawls will start in parallel.

Example: To provide the start URI for the `www.fatwire.com` site:

```
/**
 * The method is used to configure the site url which needs to
    be crawled.
 */
  public String[] getStartUri()
  {
        return ["http://www.fatwire.com/home"]; //Groovy uses
           brackets for an array.
  }
```

Example: To provide multiple start URIs for the `www.fatwire.com` site, enter a comma-separated array:

```
/**
 * The method is used to configure the site url which needs to
    be crawled.
 */
  public String[] getStartUri()
  {
        return ["http://www.fatwire.com/product","http://
           www.fatwire.com/support"]; //Groovy uses brackets
           for an array.
  }
```

createLinkExtractor

This method is used to configure the logic for extracting links from the crawled pages. The extracted links are then traversed. The `createLinkExtractor` method is a factory method in the `LinkExtractor` interface:

- You can implement the `LinkExtractor` interface to create your own link extraction algorithm – for example, using an HTML parser to parse the pages and extract links for the crawler to consume.
- You can also use the default implementation, `PatternLinkExtractor`, which uses regular expressions for extracting links. For example, `PatternLinkExtractor` can be used to extract links of the format `/home/products` from expressions such as `<a href="/home/product">Products</a>`, as shown in the sample code on page 40.

Example: To use a regular expression for extracting links from `<a href="/home/product">Products</a>` on the `www.fatwire.com` site:

```
/**
 * The method is used to define the link extraction
    algorithm from the crawled pages.
 * PatternLinkExtractor is a regex based extractor which
    parses the links on the web page
 * based on the pattern configured inside the constructor.
 *
 */
public LinkExtractor createLinkExtractor()
{
        return new PatternLinkExtractor("['\"\\(](/
           [^\\s<'\"\\)]*)",1);
}
```

- For more information about regular expressions and `PatternLinkExtractor`, see "Using the Default Implementation of LinkExtractor," on page 49.
- For more information about implementing the `LinkExtractor` interface, see "Writing and Deploying a Custom Link Extractor," on page 50.

Basic Configuration File

Below is an example of a simple `CrawlerConfigurator.groovy` file in which the required methods, `getStartUri()` and `createLinkExtractor()`, are overridden. Their code begins on line 35.

Note

In the sample below, we override an additional method `getMaxLinks()`, which is discussed on page 42. In the sample, it is set to return `150` so that the test run can be completed quickly.

The file named `CrawlerConfigurator.groovy` is used to inject dependency. Hence, its name must not be changed.

```
1     package com.fatwire.crawler.sample
2
3     import java.text.DateFormat;
4     import java.text.SimpleDateFormat;
5
6     import java.util.regex.Pattern;
7
8     import javax.mail.internet.AddressException;
9     import javax.mail.internet.InternetAddress;
10
11    import com.fatwire.crawler.*;
12    import com.fatwire.crawler.remote.*;
13    import com.fatwire.crawler.remote.di.*;
14    import com.fatwire.crawler.impl.*;
```

```
15    import com.fatwire.crawler.util.FileBuilder;
16
17    import org.apache.commons.lang.SystemUtils;
18    import org.apache.http.HttpHost;
19    import org.apache.http.auth.*;
20    import org.apache.http.client.*;
21    import org.apache.http.impl.client.*;
22    /**
23    * Configurator for the crawler.
24    * This is used to inject the dependency inside the crawler
      to control the crawling process
25    */
26
27    public class CrawlerConfigurator extends BaseConfigurator {
28
29    public CrawlerConfigurator(GlobalConfigurator delegate){
30    super(delegate);
31    }
32
33
34    /**
35    * The method is used to configure the site url which needs
      to be crawled.
36    */
37    public String[] getStartUri() {
38    return ["http://www.fatwire.com/home"]; //Groovy uses
      brackets for an array.
39    }
40
41    /**
42    * The method is used to define the link extraction algorithm
      from the crawled pages.
43    * PatternLinkExtractor is a regex based extractor which
      parses the links on the web page
44    * based on the pattern configured inside the constructor.
45    *
46    */
47    public LinkExtractor createLinkExtractor() {
48    return new PatternLinkExtractor("['\"\\(](/
      [^\\s<'\"\\)]*)",1);
49    }
50
51    /**
52    * The method is used to control the maximum number of links
      to be crawled as part
53    * of this crawl session.
54    */
55    public int getMaxLinks()
56    {
57    150;
58    }
}
```

Crawler Customization Methods

In addition to the required methods, the `BaseConfigurator` class has methods with default implementations that you may want to override to customize the crawl process in a way that agrees with the structure of the target site and the data you need to collect.

getMaxLinks

This method is used to control the number of links to be crawled. The number of links should be a positive integer. Otherwise, the crawl will scan all the links in the same domain that are reachable from the start URI(s).

Example: To specify crawling 500 links:

```
/**
 * default: -1; crawler will crawl over all the links
     reachable from the start URI
 * @return the maximum number of links to download.
 */
public int getMaxLinks()
{
     return 500;
}
```

getMaxCrawlDepth

This method is used to control the maximum depth to which a site will be crawled. Links beyond the specified depth are ignored. The depth of the starting page is `0`.

Example: To specify a crawl depth of 4:

```
/**
 * default: -1. Indicates infinite depth for a site.
 * @return the maximum depth to which we need to crawl the
     links.
 */
 public int getMaxCrawlDepth()
 {
     return 4;
 }
```

getConnectionTimeout

This method determines how long the crawler will wait to establish a connection to its target site. If a connection is not established within the specified time, the crawler will ignore the link and continue to the next link.

Example: To set a connection timeout of 50,000 milliseconds:

```
/**
 * default: 30000 ms
 * @return Connection timeout in milliseconds.
 */
 public int getConnectionTimeout()
 {
     return 50000; // in milliseconds
}
```

getSocketTimeout

This method controls the socket timeout of the request that is made by the crawler for the link to be crawled.

Example: To provide a socket timeout of 30,000 milliseconds:

```
/**
 * default: 20000 ms
 * @return Socket timeout in milliseconds.
 */
 public int getSocketTimeout()
 {
     return 30000; // in milliseconds
 }
```

getPostExecutionCommand

This method is used to inject custom post-crawl logic. This method is invoked when the crawler finishes its crawl session. The absolute path of the script or command and parameters (if any) must be returned by this method.

For example, the `getPostExecutionCommand()` can be used to automate deployment to a web server's doc base by invoking a batch or shell script to copy statically captured files after the crawl session ends.

Note

- The script or command should be present in the same location on all servers hosting Site Capture.
- Avoid downloading large archive files (exceeding 250MB) from the Site Capture interface. Use `getPostExecutionCommand` to copy the files from the Site Capture file system to your preferred location. (Archive size can be obtained from the crawler report, on the "Job Details" screen. Paths to the "Job Details" screen are shown in Figure 2, on page 33.)

Example: To execute a batch script named `copy.bat` on the Site Capture server:

```
/**
 * default: null.
 * @return the command string for post execution. Null if there
       is no such command.
*/
public String getPostExecutionCommand()
{
     // The file is supposed to be at the path C:\\commands
        folder
     // on the machine where the site capture server is running
     return "C:\\commands\\copy.bat";
}
```

getNumWorkers

This method controls the number of worker threads used for the crawl process. The ideal number of parallel threads to be spawned for the crawl session depends on the architecture of the machine on which Site Capture is hosted.

Example: To start 10 worker threads for a crawl process:

```
/**
 * default: 4.
 * @return the number of workers to start.
 * Workers will concurrently downloads resources.
 */
public int getNumWorkers()
{
     // Start 10 worker threads which is involved in the crawl
        process.
     return 10;
}
```

getUserAgent

This method is used to configure the user agent that will be utilized by the crawler when it traverses the site. This method should be used when the site is rendered differently from the way it is rendered in a browser (for example, when the site is rendered on a mobile device).

Example: To configure the FireFox 3.6.17 user agent:

```
/**
 * default: publish-crawler/1.1 (http://www.fatwire.com)
 * @return the user agent identifier
 */
public String getUserAgent()
{
     return "Mozilla/5.0 (Windows; U; Windows NT 6.1; en-
        US;rv:1.9.2.17) Gecko/20110420 Firefox/3.6.17 ";
}
```

createResourceRewriter

This method is used to rewrite URLs inside the html pages that are crawled. For example, you may want to rewrite the URLs to enable static delivery of a dynamic WebCenter Sites website.

The `createResourceRewriter` method is a factory method in the `ResourceRewriter` interface:

- You can implement the `ResourceRewriter` interface to convert dynamic URLs to static URLs, convert absolute URLs to relative URLs, and so on.
- You can also use the following default implementations:
  - `NullResourceRewriter`: Does not rewrite any of the URLs.
  - `PatternResourceRewriter`: Searches for a regular pattern and rewrites as specified.

Example: To use `PatternResourceRewriter` to rewrite URLs such as `http://www.site.com/home.html` to `/home.html`:

```
/**
 * Factory method for a ResourceRewriter.
 * default: new NullResourceRewriter();
 * @return the rewritten resource modifies the html before it
    is saved to disk.
*/
public ResourceRewriter createResourceRewriter()
{
     new PatternResourceRewriter("http://www.site.com/
        ([^\\s'\"]*)", '/$1');
}
```

- For more information about the default implementations, see "Using the Default Implementations of ResourceRewriter," on page 54.
- For more information about implementing the `ResourceRewriter` interface, see "Writing a Custom ResourceRewriter," on page 54.

createMailer

This method provides the implementation for sending email at the end of the crawl. The `createMailer` method is a factory method in the `Mailer` interface:

- Site Capture comes with an SMTP over TLS implementation, which emails the crawler report when a static or archive capture session ends (the crawler report is the `report.txt` file, described on page 31).
- If you are using a mail server other than SMTP-TLS (such as SMTP without authentication, or POP3), you will have to provide your own implementation.

Example: To send no email:

```
/**
 * Factory method for a Mailer.
 * <p/>
 * default: new NullMailer().
 * @return mailer holding configuration to send an email at
    the end of the crawl.
 * Should not be null.
 */
public Mailer createMailer()
{
     return new NullMailer();
}
```

- For more information about the default implementation, see "Using the Default Implementation of Mailer," on page 58.
- For more information about implementing the `ResourceRewriter` interface, see "Writing a Custom Mailer," on page 59.

getProxyHost

This method must be overridden if the site being crawled is behind a proxy server. You can configure the proxy server in this method.

Note

If you use `getProxyHost`, also use `getProxyCredentials`, described on page 47.

Example: To configure a proxy server:

```
/**
 * default: null.
 * @return the host for the proxy,
 * null when there is no proxy needed
 */
public HttpHost getProxyHost()
{
     //using the HTTPClient library return a HTTPHost
     return new HttpHost("www.myproxyserver.com", 883);
}
```

getProxyCredentials

This method is used to inject credentials for the proxy server that is configured in the `getProxyHost` method (page 46).

Example: To authenticate a proxy server user named `sampleuser`:

```
/**
 * default: null.
 * example: new UsernamePasswordCredentials(username,
     password);
 * @return user credentials for the proxy.
 */
public Credentials getProxyCredentials()
{
     return new UsernamePasswordCredentials("sampleuser",
        "samplepassword");
     //using the HTTPClient library return credentials
}
```

Interfaces

Site Capture provides the following interfaces with default implementations:

- LinkExtractor
- ResourceRewriter
- Mailer

LinkExtractor

A link extractor is used to specify which links will be traversed by Site Capture in a crawl session. The implementation is injected through the `CrawlerConfigurator.groovy` file. The implementation is called by the Site Capture framework during the crawl session to extract links from the markup that is downloaded as part of the crawl session.

Site Capture comes with one implementation of `LinkExtractor`. You can also write and deploy your own custom link extraction logic. For more information, see the following sections:

- LinkExtractor Interface
- Using the Default Implementation of LinkExtractor
- Writing and Deploying a Custom Link Extractor

LinkExtractor Interface

This interface has only one method (`extract`) that needs to be implemented to provide the algorithm for extracting links from downloaded markup.

```
package com.fatwire.crawler;

import java.util.List;

import com.fatwire.crawler.url.ResourceURL;


/**
 * Extracts the links out of a WebResource.
 *
 */


public interface LinkExtractor
{
     /**
      * Parses the WebResource and finds a list of links (if
             possible).
      *
      * @param resource the WebResource to inspect.
      * @return a list of links found inside the WebResource.
     */
     List<ResourceURL> extract(final WebResource resource);
     }
```

Using the Default Implementation of LinkExtractor

`PatternLinkExtractor` is the default implementation for the `LinkExtractor` interface. `PatternLinkExtractor` extracts links on the basis of a regular expression. It takes a regular expression as input and returns only links matching that regular expression.

Common usage scenarios are described below. They are:
Using `PatternLinkExtractor` for sites with dynamic URLs, and using `PatternLinkExtractor` for sites with static URLs.

- Using `PatternLinkExtractor` for sites with dynamic URLs:

  For example, on `www.fatwire.com`, the links have a pattern of `/home`, `/support`, and `/cs/Satellite/`. To extract and traverse such kinds of links, we use `PatternLinkExtractor` in the following way:

```
/**
   * The method is used to define the link extraction
     algorithm from the crawled pages.
   * PatternLinkExtractor is a regex based extractor which
     parses the links on the web page
   * based on the pattern configured inside the constructor.
   *
   */
public LinkExtractor createLinkExtractor()
{
     return new PatternLinkExtractor("['\"\\(](/
        [^\\s<'\"\\)]*)",1);
}
```

  The pattern `['\"\\(] (/[^\\s<'\"\\)]*)` is used to extract links

  - that start with any one of the following characters:
    - single quote (')
    - double quotes (")
    - opening bracket `(`
  - continue with a forward slash (/) ,
  - and end with any one of the following characters:
    - spaces (`\s`)
    - less-than symbol (<)
    - single quote (')
    - double quote (")
    - closing bracket `)`

  Let's consider the URL inside the following markup:

```
<a href='/home'>Click Me</a>
```

  We are interested only in extracting the `/home` link. This link matches the regular expression pattern because it starts with a single quote (`'`) and ends with a single quote (`'`). The grouping of 1 will return the result as `/home`.

- Using `PatternLinkExtractor` for sites with static URLs:

  For example, the markup for `www.mysite.com` has links such as:

  ```
  <a href="http://www.mysite.com/home/index.html">Click Me</a>"
  ```

  To extract and traverse such types of links, we can use `PatternLinkExtractor` in the following way:

```
/**
 * The method is used to define the link extraction algorithm
   from the crawled pages.
 * PatternLinkExtractor is a regex based extractor which
   parses the links on the web page
 * based on the pattern configured inside the constructor.
 *
 */
public LinkExtractor createLinkExtractor()
{
     return new PatternLinkExtractor(Pattern.compile("http://
        www.mysite.com/[^\\s<'\"]*"));
}
```

  The above example instructs the crawler to extract links that start with `http://www.mysite.com` and end with any one of the following characters: spaces (`\s`), less-than symbol (`<`), single quote (`'`), or double quotes (`"`).

Note

For more details on groups and patterns, refer to the Javadoc for the `Pattern` and `Matcher` classes.

Writing and Deploying a Custom Link Extractor

Note

Site Capture provides a sample link extractor (and resource rewriter) used by the FirstSiteII sample crawler to download WebCenter Sites' FirstSiteII dynamic website as a static site. For more information, see the source code for the `FSIILinkExtractor` class in the following folder:

```
<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/
  FirstSiteII/src
```

To write a custom link extractor:

1. Create a project in your favorite Java IDE.
2. Copy the file `fw-crawler-core.jar` from the `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib` folder to your project's build path.
3. Implement the `LinkExtractor` interface to provide the implementation for the `extract()` method. (The `LinkExtractor` interface is shown on page 48.)

Below is pseudo-code showing a custom implementation:

```
package com.custom.crawler;

import java.util.List;

import com.fatwire.crawler.url.ResourceURL;
import com.fatwire.crawler.LinkExtractor;

/**
 * Extracts the links out of a WebResource.
 *
 */
public class CustomLinkExtractor implements LinkExtractor
     {
     /**
      * A sample constructor for CustomLinkExtractor
      */
     public CustomLinkExtractor(String .......)
     {
           // Initialize if there are private members.
           // User's custom logic
     }

     /**
      * Parses the WebResource and finds a list of links (if
          possible).
      *
      * @param resource the WebResource to inspect.
      * @return a list of links found inside the WebResource.
      */
     List<ResourceURL> extract(final WebResource resource)
     {
           // Your custom code for extraction Algorithm.
     }

}
```

4. Create a `jar` file for your custom implementation and copy it to the following folder:
   `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib`
5. Restart your Site Capture application server.

6. Inject the dependency by coding the `CrawlerConfigurator.groovy` file to include the custom link extractor class (`CustomLinkExtractor`, in this example):

```
/*
 * User's custom link extractor mechanism to extract the
   links from the
 * web resource downloaded as part of the crawl session.
 *
 * The code below is only a pseudo code for an example.
   User is free to implement
 * their own custom constructor as shown in the next
   example.
 */
public LinkExtractor createLinkExtractor()
{
     return new CustomLinkExtractor("Custom Logic For Your
        Constructor");
}
```

ResourceRewriter

A resource rewriter is used to rewrite URLs inside the the markup that is downloaded during the crawl session. The implementation must be injected through the `CrawlerConfigurator.groovy` file.

Some use cases that will require a resource rewriter are:

- Crawling a dynamic site and creating a static copy. For example, the FirstSiteII sample site has dynamic links. Converting FirstSiteII into a static site requires rewriting the URLs inside the downloaded markup.
- Converting absolute URLs to relative URLs. For example, if the markup has URLs such as `http://www.mysite.com/abc.html`, the crawler should remove `http://www.mysite.com` from the URL, thus allowing resources to be served from the host on which the downloaded files are stored.

Site Capture comes with the two implementations of `ResourceRewriter`. You can also create custom implementations. For more information, see the following sections:

- ResourceRewriter Interface
- Using the Default Implementations of ResourceRewriter
- Writing a Custom ResourceRewriter

ResourceRewriter Interface

The `rewrite` method is used to rewrite URLs inside the markup that is downloaded during the crawl session.

```
package com.fatwire.crawler;

import java.io.IOException;


/**
 * Service for rewriting a resource. The crawler will use the
   implementation for
 * rewrite method to rewrite the resources that are downloaded as
   part of crawl
 * session.
 */
public interface ResourceRewriter
{

     /**
      * @param resource
      * @return the bytes after the rewrite.
      * @throws IOException
      */
     byte[] rewrite(WebResource resource) throws IOException;


}
```

Using the Default Implementations of ResourceRewriter

Site Capture comes with the following implementations of `ResourceRewriter`:

- `NullResourceRewriter`, configured by default to skip the rewriting of links. If `ResourceRewriter` is not configured in the `CrawlerConfigurator.groovy` file, `NullResourceRewriter` is injected by default.
- `PatternResourceRewriter`, used to rewrite URLs based on the regular expression. `PatternResourceRewriter` takes as input a regular expression to match the links inside the markup and replaces those links with the string that is provided inside the constructor.

  Example: To rewrite an absolute URL as a relative URL:

  From:

  ```
  <a href="http://www.site.com/about/index.html">Click Me</a>
  ```

  To:

  ```
  <a href="/about/index.html">Click Me</a>
  ```

  ```
  /**
   * Factory method for a ResourceRewriter.
   * default: new NullResourceRewriter();
   * @return the rewritten resource modifies the html before it
     is saved to disk.
  */
  public ResourceRewriter createResourceRewriter()
  {
       new PatternResourceRewriter("http://www.site.com/
          ([^\\s'\"]*)", '/$1');
  }
  ```

  `PatternResourceRewriter` has only one constructor that takes a regular expression and a string replacement:

  ```
  PatternResourceRewriter(final String regex, final String
     replacement)
  ```

Writing a Custom ResourceRewriter

Note

Site Capture provides a sample resource rewriter (and link extractor) used by the FirstSiteII sample crawler to download WebCenter Sites' FirstSiteII dynamic website as a static site. For more information, see the source code for the `FSIILinkExtractor` class in the following folder:

```
<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/
  FirstSiteII/src
```

To write a custom resource rewriter:

1. Create a project inside your favorite IDE.
2. Copy the files `fw-crawler-core.jar` from `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib` folder to your project's build path.
3. Implement the `ResourceRewriter` interface to provide the implementation for the `rewrite` method. (The `ResourceRewriter` interface is shown on page 53.)

   Below is pseudo-code showing a custom implementation:

```
package com.custom.crawler;

import com.fatwire.crawler.WebResource;
import com.fatwire.crawler.ResourceRewriter;


/**
 * Rewrite the links inside the markup downloaded as part of
 * crawl session.
 */
public class CustomResourceRewriter implements
     ResourceRewriter
{
     /**
      * A sample constructor for CustomResourceRewriter
      */
     public CustomResourceRewriter(String .......)
     {
           // Initialize if there are private members.
           // User's custom logic
     }

     /**
      * @param resource
      * @return the bytes after the rewrite.
      * @throws IOException
      */
     byte[] rewrite(WebResource resource) throws IOException
     {
           // Your custom code for re-writing Algorithm.
     }

}
```

4. Create a `jar` file for your custom implementation and copy it to the following folder: `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib`
5. Restart your Site Capture application server.

6. Inject the dependency by coding the `CrawlerConfigurator.groovy` file to include the custom `resource rewriter` class (`CustomResourceRewriter`, in this example):

```
/*
 * User's custom resource rewriting mechanism to rewrite
   the links from the
 * web resource downloaded as part of the crawl session.
 *
 * The code below is only a pseudo code for an example.
   User is free to implement
 * their own custom constructor as shown in the next
   example.
 */
public ResourceRewriter createResourceRewriter()
{
     new CustomResourceRewriter("User's custom logic to
        initialize the things");
}
```

Mailer

A mailer is used to send email after the crawl ends. The implementation must be injected through the `CrawlerConfigurator.groovy` file.

Site Capture provides an `SMTPTlsMailer` implementation, which can be used to send the crawler report from the SMTP-TLS mail server. You can also implement the `Mailer` interface to provide custom logic for sending emails from a server other than SMTP-TLS (such as SMTP without authentication, or POP3). Your custom logic can also specify the email to be an object other than the crawler report. If `Mailer` is not configured in the `CrawlerConfigurator.groovy` file, `NullMailer` is injected by default. For more information, see the following topics:

- Mailer Interface
- Using the Default Implementation of Mailer
- Writing a Custom Mailer

Mailer Interface

The `sendMail` method is automatically called if the `Mailer` is configured in the `CrawlerConfigurator.groovy` file.

```
package com.fatwire.crawler;

import java.io.IOException;

import javax.mail.MessagingException;


/**
 * Service to send an email.
 */
public interface Mailer
{
        /**
         * Sends the mail.
         *
         * @param subject
         * @param report
         * @throws MessagingException
         * @throws IOException
         */
        void sendMail(String subject, String report)
                throws MessagingException, IOException;
}
```

Using the Default Implementation of Mailer

Site Capture provides an SMTP-TLS server-based email implementation that sends out the crawler report when a static or archive crawl session ends (the crawler report is a (`report.txt` file, described on page 31). You can use the default mailer by injecting it through the `CrawlerConfigurator.groovy` file as shown below.

```
/**
 * Factory method for a Mailer.
 * <p/>
 * default: new NullMailer().
 * @return mailer holding configuration to send an email at the
   end of the crawl.
 * Should not be null.
 */
public Mailer createMailer()
{
     try
     {
        // Creating a SmtpTlsMailer Object
        SmtpTlsMailer mailer = new SmtpTlsMailer();

        InternetAddress from;
        // Creating an internet address from whom the mail should
           be sent from = new InternetAddress("from@gmail.com");

        // Setting the mail address inside the mailer object
           mailer.setFrom(from);

        // Setting the email address of the recipient inside
           mailer.mailer.setTo(InternetAddress.parse("xxxxxx@xxx
           xx.com"));

        // Setting the email server host for to be used for
           email.
        // The email server should be SMTP-TLS enabled.
        mailer.setHost("smtp.gmail.com", 587);

        // Setting the credentials of the mail account
           mailer.setCredentials("from@gmail.com",
           "frompassword");

        return mailer;
      }
     catch (AddressException e)
     {
        log.error(e.getMessage());
     }
}
```

Writing a Custom Mailer

To write a custom mailer:

1. Create a project inside your favorite IDE.
2. Copy the files `fw-crawler-core.jar` from the `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib` folder in your project's build path.
3. Implement the `Mailer` interface with the `sendMail` method.

   Below is pseudo-code showing a custom implementation:

```
package com.custom.crawler;

import java.io.IOException;
import javax.mail.MessagingException;
import com.fatwire.crawler.Mailer;


/**
 * Implements an interface to implement the logic for sending
   emails
 * when the crawl session has been completed.
 */
public class CustomMailer implements Mailer
{
        /**
        * A sample constructor for CustomMailer
        */
        public CustomMailer()
        {
           // Initialize if there are private members.
           // User's custom logic
        }

        /**
        * Sends the mail.
        *
        * @param subject
        * @param report
        * @throws MessagingException
        * @throws IOException
        */
        void sendMail(String subject, String report)
              throws MessagingException, IOException
        {
           // User's custom logic to send the emails.
        }

}
```

4. Create a `jar` file for your custom implementation and copy it to the following folder: `<SC_INSTALL_DIR>/fw-site-capture/webapps/ROOT/WEB-INF/lib`
5. Restart your Site Capture application server.
6. Inject the dependency by coding the `CrawlerConfigurator.groovy` file to include the custom mailer class (`CustomMailer`, in this example):

```
/**
 * Factory method for a Mailer.
 * <p/>
 * default: new NullMailer().
 * @return mailer holding configuration to send an email at
   the end of the crawl.
 * Should not be null.
 */
public Mailer createMailer()
{
        CustomMailer mailer = new CustomMailer();
        // Do some of the initilization stuffs
        return mailer;
}
package com.custom.crawler;

import java.io.IOException;
import javax.mail.MessagingException;
import com.fatwire.crawler.Mailer;


/**
 * Implements an interface to implement the logic for sending
   emails
 * when the crawl session has been completed.
 */
public class CustomMailer implements Mailer
{
        /**
         * A sample constructor for CustomMailer
         */
        public CustomMailer()
        {
           // Initialize if there are private members.
           // User's custom logic
        }
/**
```

```
         * Sends the mail.
         *
         * @param subject
         * @param report
         * @throws MessagingException
         * @throws IOException
         */
         void sendMail(String subject, String report)
                     throws MessagingException, IOException
         {
            // User's custom logic to send the emails.
         }

}
```

The implementation above will email the crawler report (`report.txt` file, described on page 31), given that the `String report` argument in the `sendMail` method names the crawler report, by default. You can customize the logic for emailing objects other than the crawler report.

Summary

This chapter discussed methods and interfaces in Site Capture's `BaseConfigurator` class for controlling a crawler's site capture process. This section summarizes the methods as well as the interfaces and their default implementations.

Methods

- getStartUri
- createLinkExtractor
- getMaxLinks
- getMaxCrawlDepth
- getConnectionTimeout
- getSocketTimeout
- getPostExecutionCommand
- getNumWorkers
- getUserAgent
- createResourceRewriter
- createMailer
- getProxyHost
- getProxyCredentials

In the preceding list, the factory methods are in the following interfaces:

- createLinkExtractor is in the LinkExtractor interface.
- createResourceRewriter is in the ResourceRewriter interface.
- createMailer is in the Mailer interface.

Interfaces

- LinkExtractor
  Its default implementation is `PatternLinkExtractor`, which extracts links on the basis of a regular expression.

  Site Capture also provides a sample link extractor (and a sample resource rewriter), used by the FirstSiteII sample crawler to download WebCenter Sites' FirstSiteII dynamic website as a static site. Source code is available in the following folder:
  `<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/FirstSiteII/src`

  You can write and deploy your own custom link extraction logic.
- ResourceRewriter
  Its default implementations are `NullResourceRewriter`, which skips the rewriting of links; and `PatternResourceRewriter`, which rewrites URLs based on the regular expression.

  Site Capture provides a sample resource rewriter (and a sample link extractor), used by the FirstSiteII sample crawler to download WebCenter Sites' FirstSiteII dynamic website as a static site. Source code is available in the following folder:
  `<SC_INSTALL_DIR>/fw-site-capture/crawler/_sample/FirstSiteII/src`

  You can write and deploy your own logic for rewriting URLs.
- Mailer
  Its default implementation is `SMTPTlsMailer`, which sends the crawler report from the SMTP-TLS mail server. You can customize the logic for emailing other types of objects from other types of servers.