# Cocoon2

## Your first Cocoon application using Maven 2

In this tutorial, you will:

* Create a Cocoon block (the application resources and logic)
* Import the block as a project in Eclipse
* Start the block as a web application and access it from your browser

## Creating a Cocoon block

Create a new directory which will be the root directory of your Cocoon application and change into it. For this tutorial, let's name it getting-started-app.The next step is to create a Cocoon block to contain your custom application. The development of any Cocoon web application should be done within one or more  blocks.  Cocoon blocks are little different to other Maven blocks except they have a particular expected [directory and file structure](http://cocoon.apache.org/2.2/core-modules/core/2.2/1263_1_1.html).

**Note:** You could manually create your blocks, but a simpler and less error-prone process is to use Maven's 'archetype' plugin.  This allows you to select from a list of block types and have Maven create them for you.  By default, the list of Maven archetypes is quite large.  For convenience, a catalog of Cocoon specific archetypes is available on the Cocoon web site and you will make use of this in the following instructions.

Run the following Maven command:

**mvn archetype:generate -DarchetypeCatalog=http://cocoon.apache.org**

Maven will generate some output and then interactively ask you for some information:

Choose archetype:

1: remote -> cocoon-22-archetype-block-plain (Creates an empty Cocoon block; useful

 if you want to add another block to a Cocoon application)

2: remote -> cocoon-22-archetype-block (Creates a Cocoon block containing some small

 samples)

3: remote -> cocoon-22-archetype-webapp (Creates a web application configured to

host Cocoon blocks. Just add the block dependencies)

Choose a number: (1/2/3): **2**

Maven first asks which archetype you want to create.  Item 2 in this list is the standard Cocoon block with some small sample files already in place.

Define value for groupId: : **com.mycompany**

Define value for artifactId: : **myBlock1**

Define value for version: 1.0-SNAPSHOT: : **1.0.0**

Define value for package: : **com.mycompany.myBlock1**

Maven then asks a few more questions.  Just enter the details as shown above.  The groupId is a label that you can use to group all the blocks in your application.  The artifactId is the unique label that will identify this block from all others.  It is also the name of the directory that will be created to contain the block's files.  The package is the Java package that any Java files in the block will be assigned to.

## Import the block in Eclipse

Change into the myBlock1 directory and call

**mvn eclipse:eclipse**

This will create the necessary project descriptors for Eclipse. In Eclipse you can import the project into your workspace: *File >  Import... > Existing Projects into Workspace*

## Run the block as Java web application

After creating the Cocoon block you probably want to run it. For this purpose there is a Maven plugin that generates a minimal web application that loads your block. The pom.xml of your block already contains the necessary settings.

**mvn jetty:run**

and point your browser at http://localhost:8888/myBlock1/ and get a hello world page

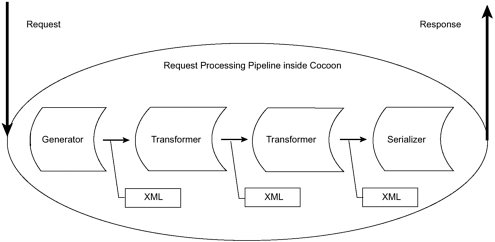
**Note:** The mentioned minimal web application is automatically created, when mvn jetty:run is invoked. This happens because the rcl goal of the Cocoon plugin is bound to the Maven build lifecycle which is invoked too, when the jetty:run goal is executed. See the block's pom.xml for details.

## Your first XML pipeline (publishing)

In this tutorial you will

* get an overview of what Cocoon pipelines and sitemaps are
*  create your first pipeline that emits XML
* enhance this first pipeline by adding an XML transformation step based on XSLT that produces HTML as output
* write a pipeline that creates a PDF document

### Overview



Cocoon is based around the concept of pipelines: If a request comes in, a [**generator**](http://cocoon.apache.org/2.2/core-modules/core/2.2/850_1_1.html) produces [SAX events](http://www.saxproject.org/) that represent the XML, [**tranformers**](http://cocoon.apache.org/2.2/core-modules/core/2.2/851_1_1.html) can alter the stream of SAX events and a [**serializer**](http://cocoon.apache.org/2.2/core-modules/core/2.2/849_1_1.html) finally creates an output stream which is sent to the client. **Sitemaps** are used to define those pipelines and connect them with requests. These connections are defined by some kind of rules. In the context  of Cocoon  those rules are named matchers. Whenever one of those rules (*matcher*) matches, a pipeline is executed.

### Create a pipeline in the sitemap

First, open src\main\resources\COB-INF\sitemap.xmap in your favorite XML editor. That's your block's base sitemap. There are already some pipelines there (/sitemap/pipelines) but for the sake of this tutorial, let's create your own pipeline.