



# Máster Universitario En Investigación En Ingeniería De Software Y Sistemas Informáticos

---

Generación Automática de Código

**Autor**

César Hugo Bárzano Cruz



TRABAJO 1 DE EVALUACIÓN CONTINUA

—  
2017/2018



# Índice general

<b>1. Introducción</b>	<b>7</b>
<b>2. Desarrollo de la Práctica</b>	<b>9</b>
2.1. Tecnologías Utilizadas . . . . .	9
2.2. Especificación de la Solución . . . . .	10
2.3. Análisis de la solución . . . . .	11
2.4. Pruebas . . . . .	11
2.4.1. Pruebas de Funcionalidad . . . . .	11
2.4.2. Pruebas de Error . . . . .	18
<b>3. Entrega</b>	<b>21</b>
3.1. Generador . . . . .	21
3.2. Makefile . . . . .	21
3.3. Directorio software specification . . . . .	21
3.4. Trabajo1_CesarHugoBarzanoCruz.pdf . . . . .	21
3.5. Directorio DOC . . . . .	22
3.6. Directorio input . . . . .	22
3.7. Directorio Output . . . . .	22
<b>4. Anexo</b>	<b>25</b>
4.1. generator.py . . . . .	25
4.2. makefile . . . . .	27



# Índice de figuras

2.1. Estructura Básica . . . . .	10
2.2. Ayuda . . . . .	10
2.3. Test Strings . . . . .	12
2.4. Test Numbers . . . . .	12
2.5. Test Objects . . . . .	12
2.6. Test Booleanos . . . . .	13
2.7. Test Null . . . . .	13
2.8. Test Arrays . . . . .	14
2.9. Test XML to Json . . . . .	18
2.10. Missing input file . . . . .	19
2.11. Bad input format . . . . .	19
2.12. Bad input json file . . . . .	20



# Capítulo 1

## Introducción

El presente documento representa la memoria formal para evaluación continua de la primera práctica de la asignatura Generación Automática de Código. Dicha práctica consiste en un generador de ficheros XML[1] como salida a partir de ficheros JSON[2] como entrada.

En los siguientes capítulos se detallará la solución propuesta, las tecnologías utilizadas, se analizarán las ventajas e inconvenientes de la solución propuesta, se explicarán los casos de uso o pruebas a los que se ha sometido el generador, se explicará como utilizar correctamente el generador así como las dependencias necesarias para su correcto uso.

Finalmente se analizarán las tareas y conocimientos adquiridos en esta práctica junto con un anexo de los documentos necesarios.





## Capítulo 2

# Desarrollo de la Práctica

### 2.1. Tecnologías Utilizadas

Tras analizar el problema propuesto, inicialmente se pensó realizar la práctica utilizando recursos Javascript debido a que las entradas del generador ( json ) están pensadas para este lenguaje, (JavaScript Object Notation) pero debido a que Javascript es una tecnología principalmente para cliente ( Front-end ) en desarrollo web, la idea de usar dicha tecnología fue rápidamente desechada.

Finalmente se ha decidido utilizar como tecnología base, el lenguaje de programación interpretado Python[3], versión 2.7.6 debido a la sencillez y capacidad del mismo para proporcionar una solución. Con el objetivo de alcanzar una implementación eficiente y no "re-inventar la rueda" se han utilizado las siguientes librerías o recursos del lenguaje con su correspondiente justificación:

1. **import json** JSON encoder and decoder Lib[4].
2. **import xmltodict** Makes working with XML feel like you are working with JSON[5].
3. **import sys** System-specific parameters and functions[6].
4. **import os** Miscellaneous operating system interfaces[7].
5. **import getopt** C-style parser for command line options[8].

Adicionalmente para la confección de esta memoria se ha utilizado LaTeX con el paquete de librerías texlive-full y el editor texmaker. Los siguientes enlaces muestran como instalar y utilizar correctamente LaTeX, han sido utilizados como referencia para el presente documento.

1. Instalar LaTeX
2. Usar LaTeX

## 2.2. Especificación de la Solución

Las posibles especificaciones e implementaciones para un generador como el propuesto por el enunciado pueden ser muy variadas. Dichas especificaciones pueden variar significativamente en función de las tecnologías utilizadas, las aserciones o comprobaciones de robustez que se quieran realizar a las entradas, al procesamiento o a las salidas, los posibles casos que se quieran tener en cuenta así como el grado de modularidad que se quiera alcanzar de cara a futuras actualizaciones. La estructura de la solución propuesta es la siguiente:

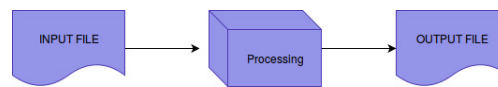


Figura 2.1: Estructura Básica

El generador recibirá como entrada un fichero Json, comprobará que dicho fichero tiene el formato adecuado (file.json), lo leerá, comprobará que el fichero json leído tiene una estructura valida (al cargarlo con la instrucción `json.loads(string)` se realiza una validación contra esquema, serializa el fichero de salida (añadiendo .xml si no se especifica el formato del fichero de salida al realizar la llamada), realizará el tratamiento, y escribirá el resultado en el fichero de salida indicado como argumento.

El generador cuenta con la siguiente ayuda con el objetivo de facilitar la comprensión de los argumentos que necesita para un uso nominal.

```

hugo@Hugo-Machine: ~/UNED/GAC/P1
hugo@Hugo-Machine:~/UNED/GAC/P1$ python generator.py -h

Usage: python generator.py [options]

Options
-v, --version          Show the version of this script
-h, --help             Show this help.
-i <path>, --input <path>  Input file
-o <path>, --output <path> Output file
-d, --debug            Debug Mode
  
```

Figura 2.2: Ayuda

Adicionalmente y debido a la fácil implementación con los recursos utilizados hasta el momento, como parte del generador se ha incluido la funcionalidad de generar ficheros JSON utilizando como entrada ficheros XML. El procedimiento es similar al funcionamiento nominal del generador, recibirá como entrada un fichero XML, comprobará que se trata de un fichero con extensión .XML, lo leerá, serializará el fichero de salida (añadiendo .json si no se ha especificado en la entrada), realizará el procesamiento y escribirá el resultado en la salida. La principal diferencia y debilidad en comparación con la funcionalidad principal ( Json to XML ) es que aquí ( XML to Json ) no se realiza una validación contra esquema para asegurar que el XML de

entrada tiene una estructura correcta. Para ello, sería necesario incluir la librería `lxml`.

Con el objetivo de facilitar la comprensión de que hace el generador en cada etapa de su ejecución, se ha incluido el flag `-d` o `--debug` el cual permite ver por la salida estándar que acción está realizando el generador desde que se invoca hasta que finaliza, se verán ejemplos de esto en la sección correspondiente a pruebas.

## 2.3. Análisis de la solución

Se considera que la solución propuesta es válida, ya que cumple con lo solicitado, realiza las adecuadas comprobaciones para una correcta ejecución y además se ha alcanzado de una manera sencilla, en un solo script python formado por una serie de funciones. La modularidad de la solución propuesta favorece a posibles actualizaciones del generador de cara al futuro, además de contar con la funcionalidad adicional de poder generar JSON desde XMLs.

Se considera interesante que para alcanzar la solución propuesta, el uso de librerías de terceros se ha limitado únicamente a lo necesario para la solución. Si bien, se podrían incluir más mecanismos y aserciones de los incluidos para garantizar a los usuarios del generador que tanto entradas como salidas tienen el formato correcto, la estructura correcta y los permisos adecuados.

## 2.4. Pruebas

Con el Objetivo de mostrar la potencia del generador, se van a realizar una serie de ejecuciones de prueba para mostrar todos los posibles escenarios del generador. Adicionalmente se va a mostrar ciertos casos de error tenidos en cuenta en la implementación de la solución, por lo que se propone dividir esta sección en las siguientes sub-secciones. Para automatizar este proceso se ha creado el `makefile` que se adjunta en el anexo donde se especifica que se va a ejecutar, tomando como entrada los ficheros alojados en el directorio `input` y dejando el resultado en el directorio `output`. En las capturas que muestran cada una de las pruebas se ve todo lo necesario para su ejecución.

### 2.4.1. Pruebas de Funcionalidad

Para mostrar la correcta funcionalidad del generador, estas pruebas van a comenzar tratando los tipos de datos básicos y una vez cubiertos, se realizarán con ficheros de entrada complejos.

#### Strings

Tipo de dato: Cadenas de caracteres

```
hugo@Hugo-Machine: ~/UNED/GAC/P1
hugo@Hugo-Machine:~/UNED/GAC/P1$ make testString
python generator.py -i input/testString.json -o output/testString.xml -d
Debug: TRUE
GAC: P1 - GENERATOR
INPUT: input/testString.json
OUTPUT: output/testString.xml
Reading input file from input/testString.json
{ "name": "Hugo" }

PROCESSING --> JSON 2 XML
Validating json input file structure...
Valid structure of input json file...
Serializing output file format...
Writing output file in output/testString.xml
<?xml version="1.0" encoding="utf-8"?>
<name>Hugo</name>
```

Figura 2.3: Test Strings

## Numbers

Tipo de dato: Números

```
hugo@Hugo-Machine: ~/UNED/GAC/P1
hugo@Hugo-Machine:~/UNED/GAC/P1$ make testNumbers
python generator.py -i input/testNumbers.json -o output/testNumbers.xml -d
Debug: TRUE
GAC: P1 - GENERATOR
INPUT: input/testNumbers.json
OUTPUT: output/testNumbers.xml
Reading input file from input/testNumbers.json
{ "age": 24 }

PROCESSING --> JSON 2 XML
Validating json input file structure...
Valid structure of input json file...
Serializing output file format...
Writing output file in output/testNumbers.xml
<?xml version="1.0" encoding="utf-8"?>
<age>24</age>
hugo@Hugo-Machine:~/UNED/GAC/P1$
```

Figura 2.4: Test Numbers

## Objects

Tipo de dato: Objetos Json

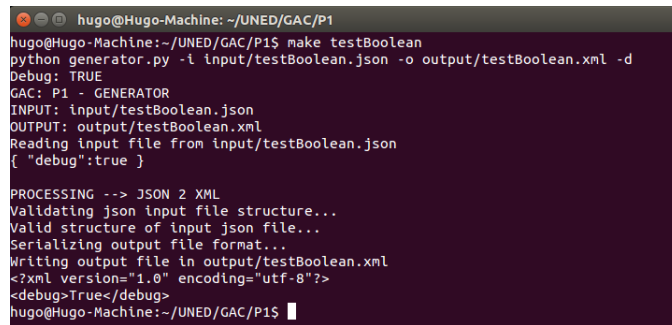
```
hugo@Hugo-Machine: ~/UNED/GAC/P1
hugo@Hugo-Machine:~/UNED/GAC/P1$ make testObjects
python generator.py -i input/testObjects.json -o output/testObjects.xml -d
Debug: TRUE
GAC: P1 - GENERATOR
INPUT: input/testObjects.json
OUTPUT: output/testObjects.xml
Reading input file from input/testObjects.json
{
  "alumno": { "name": "Hugo", "age": 24, "city": "Madrid" }
}

PROCESSING --> JSON 2 XML
Validating json input file structure...
Valid structure of input json file...
Serializing output file format...
Writing output file in output/testObjects.xml
<?xml version="1.0" encoding="utf-8"?>
<alumno>
  <city>Madrid</city>
  <age>24</age>
  <name>Hugo</name>
</alumno>
hugo@Hugo-Machine:~/UNED/GAC/P1$
```

Figura 2.5: Test Objects

## Boolean

Tipo de dato: Booleanos



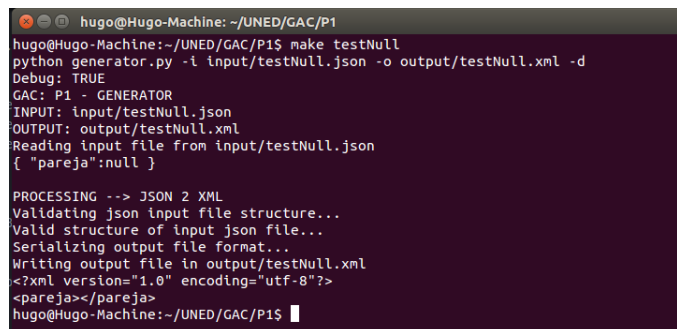
```
hugo@Hugo-Machine: ~/UNED/GAC/P1
hugo@Hugo-Machine:~/UNED/GAC/P1$ make testBoolean
python generator.py -i input/testBoolean.json -o output/testBoolean.xml -d
Debug: TRUE
GAC: P1 - GENERATOR
INPUT: input/testBoolean.json
OUTPUT: output/testBoolean.xml
Reading input file from input/testBoolean.json
{ "debug":true }

PROCESSING --> JSON 2 XML
Validating json input file structure...
Valid structure of input json file...
Serializing output file format...
Writing output file in output/testBoolean.xml
<?xml version="1.0" encoding="utf-8"?>
<debug>True</debug>
hugo@Hugo-Machine:~/UNED/GAC/P1$
```

Figura 2.6: Test Booleanos

## Null

Tipo de dato: null



```
hugo@Hugo-Machine: ~/UNED/GAC/P1
hugo@Hugo-Machine:~/UNED/GAC/P1$ make testNull
python generator.py -i input/testNull.json -o output/testNull.xml -d
Debug: TRUE
GAC: P1 - GENERATOR
INPUT: input/testNull.json
OUTPUT: output/testNull.xml
Reading input file from input/testNull.json
{ "pareja":null }

PROCESSING --> JSON 2 XML
Validating json input file structure...
Valid structure of input json file...
Serializing output file format...
Writing output file in output/testNull.xml
<?xml version="1.0" encoding="utf-8"?>
<pareja></pareja>
hugo@Hugo-Machine:~/UNED/GAC/P1$
```

Figura 2.7: Test Null

## Arrays

Tipo de dato: Listas

```

hugo@Hugo-Machine: ~/UNED/GAC/P1
hugo@Hugo-Machine:~/UNED/GAC/P1$ make testArrays
python generator.py -i input/testArrays.json -o output/testArrays.xml -d
Debug: TRUE
GAC: P1 - GENERATOR
INPUT: input/testArrays.json
OUTPUT: output/testArrays.xml
Reading input file from input/testArrays.json
{
  "planets": {
    "planet": [{"name": "Earth"}, {"name": "Jupiter"}, {"name": "Mars"}]
  }
}

PROCESSING --> JSON 2 XML
Validating json input file structure...
Valid structure of input json file...
Serializing output file format...
Writing output file in output/testArrays.xml
<?xml version="1.0" encoding="utf-8"?>
<planets>
  <planet>
    <name>Earth</name>
  </planet>
  <planet>
    <name>Jupiter</name>
  </planet>
  <planet>
    <name>Mars</name>
  </planet>
</planets>
hugo@Hugo-Machine:~/UNED/GAC/P1$

```

Figura 2.8: Test Arrays

### Json complejos

La siguiente prueba muestra la capacidad del generador para trabajar con ficheros json complejos, como es por ejemplo el servlet de una aplicación web. Debido a la extensión de la salida de esta prueba, no se adjuntará captura, si no la salida estándar.

Listing 2.1: make testServlet

```

1 hugo@Hugo-Machine:~/UNED/GAC/P1$ make testServlet
2 python generator.py -i input/servlet.json -o output/output_testervlet
3 .xml -d
4 Debug: TRUE
5 GAC: P1 - GENERATOR
6 INPUT: input/servlet.json
7 OUTPUT: output/output_testervlet.xml
8 Reading input file from input/servlet.json
9 {"web-app": {
10   "servlet": [
11     {
12       "servlet-name": "cofaxCDS",
13       "servlet-class": "org.cofax.cds.CDSServlet",
14       "init-param": {
15         "templateProcessorClass": "org.cofax.WysiwygTemplate",
16         "templateLoaderClass": "org.cofax.FilesTemplateLoader",
17         "templatePath": "templates",
18         "templateOverridePath": "",
19         "defaultListTemplate": "listTemplate.htm",
20         "defaultFileTemplate": "articleTemplate.htm",
21         "useJSP": false,
22         "jspListTemplate": "listTemplate.jsp",
23         "jspFileTemplate": "articleTemplate.jsp",
24         "cachePackageTagsTrack": 200,
25         "cachePackageTagsStore": 200,

```

```
25     "cachePackageTagsRefresh": 60,
26     "cacheTemplatesTrack": 100,
27     "cacheTemplatesStore": 50,
28     "cacheTemplatesRefresh": 15,
29     "cachePagesTrack": 200,
30     "cachePagesStore": 100,
31     "cachePagesRefresh": 10,
32     "cachePagesDirtyRead": 10,
33     "searchEngineListTemplate": "forSearchEnginesList.htm",
34     "searchEngineFileTemplate": "forSearchEngines.htm",
35     "searchEngineRobotsDb": "WEB-INF/robots.db",
36     "useDataStore": true,
37     "dataStoreClass": "org.cofax.SqlDataStore",
38     "redirectionClass": "org.cofax.SqlRedirection",
39     "dataStoreName": "cofax",
40     "dataStoreDriver": "com.microsoft.jdbc.sqlserver.
    SQLServerDriver",
41     "dataStoreUrl": "jdbc:microsoft:sqlserver://LOCALHOST:1433;
    DatabaseName=goon",
42     "dataStoreUser": "sa",
43     "dataStorePassword": "dataStoreTestQuery",
44     "dataStoreTestQuery": "SET NOCOUNT ON;select test='test';",
45     "dataStoreLogFile": "/usr/local/tomcat/logs/datastore.log",
46     "dataStoreInitConns": 10,
47     "dataStoreMaxConns": 100,
48     "dataStoreConnUsageLimit": 100,
49     "dataStoreLogLevel": "debug",
50     "maxUrlLength": 500}},
51 {
52     "servlet-name": "cofaxEmail",
53     "servlet-class": "org.cofax.cds.EmailServlet",
54     "init-param": {
55         "mailHost": "mail1",
56         "mailHostOverride": "mail2"}}},
57 {
58     "servlet-name": "cofaxAdmin",
59     "servlet-class": "org.cofax.cds.AdminServlet"},
60
61 {
62     "servlet-name": "fileServlet",
63     "servlet-class": "org.cofax.cds.FileServlet"},
64 {
65     "servlet-name": "cofaxTools",
66     "servlet-class": "org.cofax.cms.CofaxToolsServlet",
67     "init-param": {
68         "templatePath": "toolstemplates/",
69         "log": 1,
70         "logLocation": "/usr/local/tomcat/logs/CofaxTools.log",
71         "logMaxSize": "",
72         "dataLog": 1,
73         "dataLogLocation": "/usr/local/tomcat/logs/dataLog.log",
74         "dataLogMaxSize": "",
75         "removePageCache": "/content/admin/remove?cache=pages&id=",
76         "removeTemplateCache": "/content/admin/remove?cache=templates&
    id=",
77         "fileTransferFolder": "/usr/local/tomcat/webapps/content/
    fileTransferFolder",
78         "lookInContext": 1,
79         "adminGroupID": 4,
80         "betaServer": true}}},
81 "servlet-mapping": {
82     "cofaxCDS": "/",
```

```

83     "cofaxEmail": "/cofaxutil/aemail/*",
84     "cofaxAdmin": "/admin/*",
85     "fileServlet": "/static/*",
86     "cofaxTools": "/tools/*"},
87
88     "taglib": {
89         "taglib-uri": "cofax.tld",
90         "taglib-location": "/WEB-INF/tlds/cofax.tld"}}}
91
92 PROCESSING --> JSON 2 XML
93 Validating json input file structure...
94 Valid structure of input json file...
95 Serializing output file format...
96 Writing output file in output/output_testervlet.xml
97 <?xml version="1.0" encoding="utf-8"?>
98 <web-app>
99     <servlet-mapping>
100         <cofaxTools>/tools/*</cofaxTools>
101         <cofaxCDS>/</cofaxCDS>
102         <fileServlet>/static/*</fileServlet>
103         <cofaxAdmin>/admin/*</cofaxAdmin>
104         <cofaxEmail>/cofaxutil/aemail/*</cofaxEmail>
105     </servlet-mapping>
106     <taglib>
107         <taglib-location>/WEB-INF/tlds/cofax.tld</taglib-
            location>
108         <taglib-uri>cofax.tld</taglib-uri>
109     </taglib>
110     <servlet>
111         <servlet-name>cofaxCDS</servlet-name>
112         <init-param>
113             <cachePagesStore>100</cachePagesStore>
114             <searchEngineListTemplate>forSearchEnginesList
                .htm</searchEngineListTemplate>
115             <maxUrlLength>500</maxUrlLength>
116             <dataStoreTestQuery>SET NOCOUNT ON;select test
                ='test';</dataStoreTestQuery>
117             <defaultFileTemplate>articleTemplate.htm</
                defaultFileTemplate>
118             <dataStoreLogFile>/usr/local/tomcat/logs/
                datastore.log</dataStoreLogFile>
119             <templateLoaderClass>org.cofax.
                FilesTemplateLoader</templateLoaderClass>
120             <dataStoreClass>org.cofax.SqlDataStore</
                dataStoreClass>
121             <templateOverridePath></templateOverridePath>
122             <cacheTemplatesStore>50</cacheTemplatesStore>
123             <dataStoreUrl>jdbc:microsoft:sqlserver://
                LOCALHOST:1433;DatabaseName=goon</
                dataStoreUrl>
124             <searchEngineFileTemplate>forSearchEngines.htm
                </searchEngineFileTemplate>
125             <cachePagesTrack>200</cachePagesTrack>
126             <cachePackageTagsStore>200</
                cachePackageTagsStore>
127             <dataStoreName>cofax</dataStoreName>
128             <dataStorePassword>dataStoreTestQuery</
                dataStorePassword>
129             <useJSP>False</useJSP>
130             <defaultListTemplate>listTemplate.htm</
                defaultListTemplate>
131             <dataStoreUser>sa</dataStoreUser>

```



```

132         <jspListTemplate>listTemplate.jsp</
        jspListTemplate>
133     <jspFileTemplate>articleTemplate.jsp</
        jspFileTemplate>
134     <dataStoreMaxConns>100</dataStoreMaxConns>
135     <cachePagesDirtyRead>10</cachePagesDirtyRead>
136     <cachePagesRefresh>10</cachePagesRefresh>
137     <cacheTemplatesTrack>100</cacheTemplatesTrack>
138     <dataStoreConnUsageLimit>100</
        dataStoreConnUsageLimit>
139     <redirectionClass>org.cofax.SqlRedirection</
        redirectionClass>
140     <searchEngineRobotsDb>WEB-INF/robots.db</
        searchEngineRobotsDb>
141     <templateProcessorClass>org.cofax.
        WysiwygTemplate</templateProcessorClass>
142     <cachePackageTagsRefresh>60</
        cachePackageTagsRefresh>
143     <templatePath>templates</templatePath>
144     <useDataStore>True</useDataStore>
145     <cacheTemplatesRefresh>15</
        cacheTemplatesRefresh>
146     <dataStoreDriver>com.microsoft.jdbc.sqlserver.
        SQLServerDriver</dataStoreDriver>
147     <cachePackageTagsTrack>200</
        cachePackageTagsTrack>
148     <dataStoreLogLevel>debug</dataStoreLogLevel>
149     <dataStoreInitConns>10</dataStoreInitConns>
150 </init-param>
151 <servlet-class>org.cofax.cds.CDSServlet</servlet-class>
        >
152 </servlet>
153 <servlet>
154     <servlet-name>cofaxEmail</servlet-name>
155     <init-param>
156         <mailHostOverride>mail2</mailHostOverride>
157         <mailHost>mail1</mailHost>
158     </init-param>
159     <servlet-class>org.cofax.cds.EmailServlet</servlet-
        class>
160 </servlet>
161 <servlet>
162     <servlet-name>cofaxAdmin</servlet-name>
163     <servlet-class>org.cofax.cds.AdminServlet</servlet-
        class>
164 </servlet>
165 <servlet>
166     <servlet-name>fileServlet</servlet-name>
167     <servlet-class>org.cofax.cds.FileServlet</servlet-
        class>
168 </servlet>
169 <servlet>
170     <servlet-name>cofaxTools</servlet-name>
171     <init-param>
172         <logLocation>/usr/local/tomcat/logs/CofaxTools
            .log</logLocation>
173         <fileTransferFolder>/usr/local/tomcat/webapps/
            content/fileTransferFolder</
            fileTransferFolder>
174         <log>1</log>
175         <dataLog>1</dataLog>
176         <dataLogLocation>/usr/local/tomcat/logs/

```

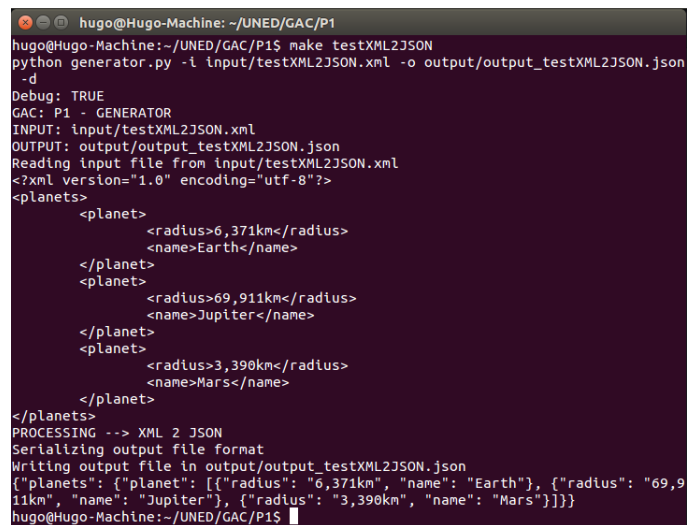
```

177         dataLog.log</dataLogLocation>
178         <adminGroupID>4</adminGroupID>
179         <lookInContext>1</lookInContext>
180         <removePageCache>/content/admin/remove?cache=
            pages&id=</removePageCache>
181         <removeTemplateCache>/content/admin/remove?
            cache=templates&id=</
            removeTemplateCache>
182         <logMaxSize></logMaxSize>
183         <dataLogMaxSize></dataLogMaxSize>
184         <betaServer>True</betaServer>
185         <templatePath>toolstemplates/</templatePath>
186     </init-param>
187     <servlet-class>org.cofax.cms.CofaxToolsServlet</
        servlet-class>
188 </servlet>
189 </web-app>
hugo@Hugo-Machine: ~/UNED/GAC/P1$

```

## XML to JSon

Como parte de la funcionalidad añadida, se va a realizar una única prueba, en la que el generador va a ingestar un fichero XML y va a generar un fichero JSON.



```

hugo@Hugo-Machine: ~/UNED/GAC/P1$
hugo@Hugo-Machine:~/UNED/GAC/P1$ make testXML2JSON
python generator.py -i input/testXML2JSON.xml -o output/output_testXML2JSON.json
-d
Debug: TRUE
GAC: P1 - GENERATOR
INPUT: input/testXML2JSON.xml
OUTPUT: output/output_testXML2JSON.json
Reading input file from input/testXML2JSON.xml
<?xml version="1.0" encoding="utf-8"?>
<planets>
  <planet>
    <radius>6,371km</radius>
    <name>Earth</name>
  </planet>
  <planet>
    <radius>69,911km</radius>
    <name>Jupiter</name>
  </planet>
  <planet>
    <radius>3,390km</radius>
    <name>Mars</name>
  </planet>
</planets>
PROCESSING --> XML 2 JSON
Serializing output file format
Writing output file in output/output_testXML2JSON.json
{"planets": [{"planet": [{"radius": "6,371km", "name": "Earth"}, {"radius": "69,911km", "name": "Jupiter"}, {"radius": "3,390km", "name": "Mars"}]}]}
hugo@Hugo-Machine:~/UNED/GAC/P1$

```

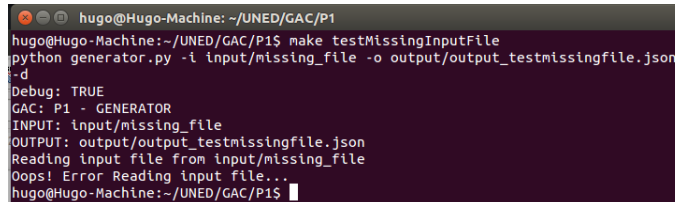
Figura 2.9: Test XML to Json

### 2.4.2. Pruebas de Error

Para realizar las pruebas de error, se ejercitarán los casos en los que las entradas no tengan el formato adecuado o la estructura del json de entrada no sea adecuada.

## Entrada inexistente

Si el usuario intenta usar como entrada ficheros que no existen, el generador notificará el error y parará:

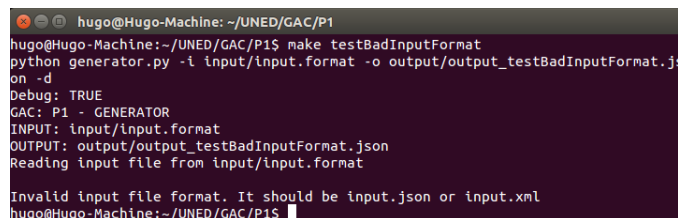


```
hugo@Hugo-Machine: ~/UNED/GAC/P1
hugo@Hugo-Machine:~/UNED/GAC/P1$ make testMissingInputFile
python generator.py -i input/missing_file -o output/output_testmissingfile.json
-d
Debug: TRUE
GAC: P1 - GENERATOR
INPUT: input/missing_file
OUTPUT: output/output_testmissingfile.json
Reading input file from input/missing_file
Oops! Error Reading input file...
hugo@Hugo-Machine:~/UNED/GAC/P1$
```

Figura 2.10: Missing input file

## Input.format

Si el usuario del generador intenta usar como entrada ficheros cuya extensión no sea .json o .xml el generador notificará el error y parará:



```
hugo@Hugo-Machine: ~/UNED/GAC/P1
hugo@Hugo-Machine:~/UNED/GAC/P1$ make testBadInputFormat
python generator.py -i input/input.format -o output/output_testBadInputFormat.js
on -d
Debug: TRUE
GAC: P1 - GENERATOR
INPUT: input/input.format
OUTPUT: output/output_testBadInputFormat.json
Reading input file from input/input.format

Invalid input file format. It should be input.json or input.xml
hugo@Hugo-Machine:~/UNED/GAC/P1$
```

Figura 2.11: Bad input format

## Estructura Json

Si el usuario del generador intenta usar como entrada un fichero json cuya estructura no sea la adecuada, el generador notificará el error y parará ya que realiza un chequeo contra esquema solo para los json de entrada.

A terminal window with a dark background and light text. The prompt is 'hugo@Hugo-Machine: ~/UNED/GAC/P1'. The user has run 'python generator.py -i input/badInput.json -o output/output\_testBADInput.xml -d'. The program output shows it's a generator, reads the input file, and displays a JSON object for planets. The JSON is: {'planets': [{'planet': {'name': 'Earth', 'radius': '6,371km'}}, {'name': 'Jupiter', 'radius': '69,911km'}], 'name': 'Mars', 'radius': '3,390km'}]. The program then says 'PROCESSING --> JSON 2 XML' and 'Validating json input file structure...'. It then outputs an error: 'Oops! Invalid structure of input json file...'.

```
hugo@Hugo-Machine: ~/UNED/GAC/P1
hugo@Hugo-Machine:~/UNED/GAC/P1$ make testBadJson
python generator.py -i input/badInput.json -o output/output_testBADInput.xml -d
Debug: TRUE
GAC: P1 - GENERATOR
INPUT: input/badInput.json
OUTPUT: output/output_testBADInput.xml
Reading input file from input/badInput.json
{
  "planets": {
    "planet": {
      {
        "name": "Earth",
        "radius": "6,371km"
      },
      {
        "name": "Jupiter",
        "radius": "69,911km"
      },
      "name": "Mars",
      "radius": "3,390km"
    }
  }
}
PROCESSING --> JSON 2 XML
Validating json input file structure...
Oops! Invalid structure of input json file...
hugo@Hugo-Machine:~/UNED/GAC/P1$
```

Figura 2.12: Bad input json file

## Capítulo 3

# Entrega

En este capítulo se detallan cada uno de los ficheros/directorios que forman parte de la entrega.

### 3.1. Generador

El generador propuesto por el enunciado se compone de un único fichero denominado `generator.py` el cual se encuentra alojado en la raíz del fichero comprimido que se envía. El generador ha sido desarrollado sobre Ubuntu14 y hace uso de la versión python 2.7.6 la cual incluye todas las dependencias necesarias. Si alguna de ellas causará error, podría ser instalada fácilmente descargando el paquete de las URLs indicadas en la bibliografía y ejecutando `sudo pip install nombre_paquete`

### 3.2. Makefile

El fichero "makefile" establece las distintas ejecuciones de pruebas para el generador. También dispone de la entrada "make doc" para generar la especificación software mediante pydoc. Alojado en la raíz del fichero comprimido para la entrega.

### 3.3. Directorio software specification

Contiene la especificación del generador en formato HTML. Dicha especificación ha sido generada utilizando la librería pydoc.

### 3.4. Trabajo1\_CesarHugoBarzanoCruz.pdf

Memoria de la práctica, referencia a este documento en si mismo, alojado en el directorio raíz de la entrega.

### **3.5. Directorio DOC**

Directorio donde se almacenan todos los fuentes usados para generar esta documentación utilizando LaTeX. Incluye tambien las imagenes usadas en la memoria.

### **3.6. Directorio input**

Directorio donde se almacenan todos los ficheros usados como entradas en la sección de pruebas

### **3.7. Directorio Output**

Directorio donde se almacenan todos los ficheros resultados de la ejecución de la pruebas.

# Bibliografía

- [1] XML, *XML Format* <https://www.w3.org/XML/>
- [2] JSON, *JSON Format* <https://www.json.org/>
- [3] PYTHON, *Python 2.7.6 language* <https://www.python.org/download/releases/2.7.6/>
- [4] IMPORT JSON, *JSON encoder and decoder* <https://docs.python.org/2/library/json.html>
- [5] IMPORT XMLTODIC, *Makes working with XML feel like you are working with JSON* <https://pypi.python.org/pypi/xmltodict>
- [6] IMPORT SYS, *System-specific parameters and functions* <https://docs.python.org/2/library/sys.html>
- [7] IMPORT OS, *Miscellaneous operating system interfaces* <https://docs.python.org/2/library/os.html>
- [8] IMPORT GETOPT, *C-style parser for command line options* <https://docs.python.org/2/library/getopt.html>





# Capítulo 4

## Anexo

### 4.1. generator.py

Listing 4.1: generator.py

```
1  """XML generator from file.json"""
2
3
4  __author__ = 'Hugo Barzano'
5  __date__ = '2017/2018'
6  __version__ = 'v1.0'
7  __credits__ = 'GAC'
8  __file__ = 'generator.py'
9
10 import json
11 import xmldict
12 import sys
13 import os
14 import getopt
15
16 def readInput(input_file):
17     """Function to read input files.
18         :param input_file: input file path"""
19     print "Reading input file from "+input_file
20     try:
21         with open(input_file, 'r') as f:
22             input_file_as_string = f.read()
23         return input_file_as_string
24     except EnvironmentError:
25         print("Oops! Error Reading input file...")
26         exit()
27
28 def writeOutput(output_file,output_string):
29     """Function to write output files.
30         :param output_file: output file path
31         :param output_string: string to write in the output file"""
32     print "Writing output file in "+output_file
33     try:
34         with open(output_file, 'w') as f:
35             f.write(output_string)
36     except EnvironmentError:
37         print("Oops! Error writing output file...")
```

```

38         exit()
39
40 def validateJson(input_string):
41     """Function to validate json inputs.
42     :param input_string: json as string to validate"""
43     try:
44         validate_input=json.loads(input_string)
45         print "Valid structure of input json file..."
46     except ValueError:
47         print("Oops! Invalid structure of input json file...")
48         exit()
49
50 def parseInputJsonToXML(input_string):
51     """Function to parse json files as input to XML string.
52     :param input_string: input data file as string """
53     xmlString = xmltodict.unparse(json.loads(input_string), pretty=
54         True)
55     return xmlString
56
57 def parseInputXMLToJson(input_string):
58     """Function to parse xml files as input to json string.
59     :param input_string: input data file as string """
60     jsonString = json.dumps(xmltodict.parse(input_string))
61     return jsonString
62
63 def usage():
64     """Method to display generator usage. """
65     print """
66 Usage: python generator.py [options]
67
68 Options
69 -v, --version          Show the version of this script
70 -h, --help             Show this help.
71 -i <path>, --input <path>   Input file
72 -o <path>, --output <path>   Output file
73 -d, --debug            Debug Mode
74 """
75
76 def version():
77     """Function to display software version"""
78     print "version 1.0"
79
80 def main(argv):
81     """Main method to execute the generator.
82     :param argv: value from 0 to n-1 to ident"""
83     input_file=None
84     output_file=None
85     debug=False
86
87     # process arguments
88     try:
89         opts, args = getopt.getopt(argv, "hvi:o:d", ["help", "version",
90             "input=", "output=", "debug"])
91     except getopt.GetoptError as err:
92         print str(err)
93         usage()
94         sys.exit(2)
95     for opt, arg in opts:
96         if opt in ("-h", "--help"):
97             usage()
98             sys.exit()
99         elif opt in ("-v", "--version"):

```

```

98         version()
99         sys.exit()
100     elif opt in ("-i", "--input"):
101         input_file = arg
102     elif opt in ("-o", "--output"):
103         output_file=arg
104     elif opt in ("-d", "--debug"):
105         debug = True
106         print "Debug: TRUE"
107     else:
108         usage()
109         sys.exit()
110
111     print "GAC: P1 - GENERATOR"
112     if debug:
113         print "INPUT: "+ input_file
114         print "OUTPUT: "+ output_file
115
116
117
118     input_as_string=readInput(input_file)
119     if debug: print input_as_string
120     if input_file.split(".")[1]=="json":
121         print "PROCESSING --> JSON 2 XML "
122         if debug: print "Validating json input file structure..."
123         validateJson(input_as_string)
124         if debug: print "Serializing output file format..."
125         if not ".xml" in output_file: output_file=output_file+".xml"
126         output_as_string=parseInputJsonToXML(input_as_string)
127     elif input_file.split(".")[1]=="xml":
128         print "PROCESSING --> XML 2 JSON"
129         if debug: print "Serializing output file format"
130         if not ".json" in output_file: output_file=output_file+".json"
131         output_as_string=parseInputXMLToJson(input_as_string)
132     else:
133         print "Invalid input file format. It should be input.json or
134             input.xml "
135         exit()
136     writeOutput(output_file,output_as_string)
137     if debug:print output_as_string
138
139 #
140 # -----
141 #
142
143 if __name__ == "__main__":
144     if len(sys.argv) > 1:
145         main(sys.argv[1:])
146     else:
147         usage()
148         sys.exit()

```

## 4.2. makefile

Listing 4.2: makefile

```
1 #Makefile
2
3 doc:
4     rm -r software_specification && mkdir software_specification
5     && pydoc -w ./ && mv ./*.html software_specification/
6
7 testString:
8     python generator.py -i input/testString.json -o output/
9     testString.xml -d
10
11 testNumbers:
12     python generator.py -i input/testNumbers.json -o output/
13     testNumbers.xml -d
14
15 testObjects:
16     python generator.py -i input/testObjects.json -o output/
17     testObjects.xml -d
18
19 testBoolean:
20     python generator.py -i input/testBoolean.json -o
21     output/testBoolean.xml -d
22
23 testNull:
24     python generator.py -i input/testNull.json -o output/
25     testNull.xml -d
26
27 testArrays:
28     python generator.py -i input/testArrays.json -o output/
29     testArrays.xml -d
30
31 testServlet:
32     python generator.py -i input/servlet.json -o output/
33     output_testervlet.xml -d
34
35 testXML2JSON:
36     python generator.py -i input/testXML2JSON.xml -o
37     output/output_testXML2JSON.json -d
38
39 testBadInputFormat:
40     python generator.py -i input/input.format -o output/
41     output_testBadInputFormat.xml -d
42
43 testMissingInputFile:
44     python generator.py -i input/missing_file -o output/
45     output_testmissingfile.xml -d
46
47 testBadJson:
48     python generator.py -i input/badInput.json -o output/
49     output_testBADInput.xml -d
```