



Projman user manual

Projman user manual

Sandrine Ribeau

2004/04/26

Copyright 2004 by Logilab

Abstract

FIXME (changed the meaning of projman/project/task elements) This is the reference manual for the Projman software. Projman has three functionalities. First, Projman reads a project description from a xml formatted file and outputs different kind of diagrams like, a Gantt diagram, or resources activities description in different formats (html, svg, png...). Second, Projman has the ability to allow data exchange between other soft like [Planner](http://mrproject.codefactory.se) [http://mrproject.codefactory.se]. And finally, Projman can schedule a project, so proposes a planned project. The main difference between Projman and other similar project is its ability to schedule the project for you, trying to satisfy all constraints given in your project description and resources availability description.

Table of Content

Chapter I - Installation	1
1. Downloads	1
2. Install	1
3. Contribute!	1
Chapter II - The projman files	2
1. Projman file	2
2. Project file	2
2.1. Project	2
2.2. Milestone	3
2.3. Task	3
3. Constraints : date constraint, task constraint and resource constraint.	4
3.1. Date constraints	4
3.2. Task constraints	4
3.3. Resource constraints	4
4. Resources file	4
4.1. Resources-list	4
4.2. Resource	5
4.3. Calendar	5
5. Scheduling file	6
6. Activities file	7
Chapter III - Converting, scheduling and rendering	8
1. Converting	8
2. Scheduling	9
2.1. Default scheduling	9
2.2. Constraint scheduling	9
3. Rendering	9
4. XML documents	10
Appendix A - Annexes	11
1. Related programs	11

Chapter I

Installation

1. Downloads

First of all, since Projman is a Python script, you must have [Python](http://www.python.org) installed !

Projman requires the [mx.DateTime](http://www.egenix.com/files/python/eGenix-mx-Extensions.html) package.

Projman itself may be found [here](ftp://ftp.logilab.org/pub/projman). The package contains the Projman libraries and a command line tool, an example, the [strptime.py](http://www.fukt.hk-r.se/~flognat/hacks/strptime.py) module for M\$ Windows unfortunate users and this documentation page. The main site is at [LOGILAB](http://www.logilab.org/projects/projman).

If you want to obtain another rendering than HTML or SVG, you must download the Pythonware's Python Imaging Library (PIL) available [here](http://www.pythonware.com/products/pil/).

Note for happy debian users: Projman is available as a debian package. You can retrieve it by adding "deb ftp://ftp.logilab.org/pub/debian unstable" to your `/etc/apt/sources.list` file.

2. Install

Projman is a Python script and provides distutils facilities. To install it, just extract it, enter in the extracted directory and run `python setup.py install`. It should be the same thing for all dependency packages.

3. Contribute!

Please consider contributing anything useful to this package.

Chapter II

The projman files

You will find here a description of the different files that maybe found in Projman description. Indeed, Projman describes at least three files: one for the project description, one for the resources description and last for the activities (past or future) description. This is the minimal requirements for Projman description. The other component that can compose Projman is the result of scheduling. All the files used for Projman description are independant from each other.

1. Projman file

```
<!ELEMENT projman (import-tasks | import-activities |
                    import-resources | import-schedule)*>
<!ELEMENT import-tasks EMPTY>
<!ATTLIST import-tasks
    file          CDATA          #REQUIRED>
<!ELEMENT import-activities EMPTY>
<!ATTLIST import-activities
    file          CDATA          #REQUIRED>

<!ELEMENT import-schedule EMPTY>
<!ATTLIST import-schedule
    file          CDATA          #REQUIRED>
<!ELEMENT import-resources EMPTY>
<!ATTLIST import-resources
    file          CDATA          #REQUIRED>
```

Example 1 - The DTD for the Projman element

The **<projman>** node is the root of a Projman description. You can import a resources file by adding a **<import-resources>** child to this node which has the file attribute set to the file name. You can import too, an activities file, a project file, and finally a schedule file by adding a **<import-activities>**, **<import-tasks>** or **<import-schedule>** child to this node with as file name, the file attribute.

2. Project file

2.1. Project

```
<!ELEMENT task (label?,resource*,(constraint-date |
                                constraint-task |constraint-resource)*,
                (task|import-project|milestone)+, priority?)>
<!ATTLIST project
    id          ID          #REQUIRED>

<!-- project from file -->
<!ELEMENT import-project EMPTY>
<!ATTLIST import-project
    file          CDATA          #REQUIRED>
```

Example 2 - Portion of the DTD for the **<project>** element

The **<project>** node is the root of a project file description.

The **<import-tasks>** node allows the project to be composed of multiple other projects. Everything will work as if this node was replaced by the content of this imported project file.

The **<constraint-date>**, **<constraint-task>** and **<constraint-resource>** nodes are all the possible constraints that can be set on a project. See [Section 3 “Constraints : date constraint, task constraint and resource constraint.” - Chapter II](#) for how to describe constraints.

2.2. Milestone

```
<!ELEMENT milestone (label?, (constraint-date |
                             constraint-task |constraint-resource)*,
                             (task|import-project|milestone)+)*>
<!ATTLIST milestone
    id                ID                #REQUIRED >
```

Example 3 - Portion of the DTD for the **<milestone>** element

The **<milestone>** node is the atomic element that can compose a project. This is related to an events which takes place at a specific date but without any duration or timing information. Task and Project node inherits from Milestone.

2.3. Task

```
<!ELEMENT task (label?, (constraint-date | constraint-task
                        |constraint-resource)*,(duration,progress?),
                priority?,(task|milestone|import-project)*)>
<!ATTLIST task
    id                ID                #REQUIRED>
<!-- label, duration and progress of the task -->
<!ELEMENT label (#PCDATA)>
<!ELEMENT duration (#PCDATA)>
<!ATTLIST duration
    unit (days|weeks|months) "days">
<!ELEMENT progress (#PCDATA)>
```

Example 4 - Portion of the DTD for the **<task>** element

The **<task>** node describes a given task, identified by its "id" attribute. A task maybe a container (i.e. contains some other tasks and/or milestone and/or imported projects) or a leaf. If it's a leaf, the **<duration>** node is required to indicate the number of days of work needed by this task. You can optionally add an "unit" attribute if you want to indicate a number of weeks or months.

The **<progress>** gives the percentage of work done for this task. A leaf task is considered as finished when its progress is 100%. Note that if you have a task with 2 days as duration and a 50% progress, Projman will consider that this task needs one more day of work.

The node **<priority>** has value in range from 0 (lowest) to 9. By default, all tasks are considered as if they had a 5 priority. You can ordering task by setting this attribute. If a task t1 has a lower priority than a task t2, it's equivalent to having a task constraint "t1 begin after begin of t2". Note that if no "priority" attribute is present, a task will inherit from its parent priority.

See [Section 3 “Constraints : date constraint, task constraint and resource constraint.” - Chapter II](#) for how to describing constraints. Another way to give constraint between task is to use the task's "priority" attribute.

3. Constraints : date constraint, task constraint and resource constraint.

```
<!-- constraint between tasks -->
<!ELEMENT constraint-task EMPTY>
<!-- ATTLIST constraint-task
      idref      IDREF          #REQUIRED
      type       (begin-after-end|begin-after-begin|
                  end-after-end|end-after-begin) #REQUIRED>

<!-- constraint on date for a task -->
<!ELEMENT constraint-date (#PCDATA)>
<!-- ATTLIST constraint-date
      type       (begin-after-date|begin-at-date|begin-before-date|
                  end-after-date|end-at-date|end-before-date)
                  #REQUIRED>

<!-- constraint on resources to use for tasks -->
<!ELEMENT constraint-resource EMPTY>
<!-- ATTLIST constraint-resource
      type       CDATA          #IMPLIED"
      usage      CDATA          #REQUIRED">
```

Example 5 - Portion of the DTD for the **<constraint-date>** **<constraint-task>** and **<constraint-resource>** element

3.1. Date constraints

The possible date constraint are the following : begin before date, begin at date, begin after date, end before date, end at date and end after date. The value of the **<constraint-date>** element must be in the standard date format.

3.2. Task constraints

A task constraint permits to give an order between tasks execution. The following type are possible : begin after end, begin after begin, end after begin and end after end. The most used is the begin after end constraint. The value of the **<constraint-task>** element must be a task id.

3.3. Resource constraints

A resource constraint permits to assign one or more resource to a task. The attribute "type" permits to design the profile of the resources needed to complete the task. Indeed "type" describes the type of the resource. The attribute "usage" gives the percentage of time the resource has to give to the task.

4. Resources file

4.1. Resources-list

```
<!ELEMENT resources-list (resource|calendar)+ >
<!-- ATTLIST resources-list
```


id	ID	#IMPLIED>
----	----	-----------

Example 6 - Portion of the DTD for the <resources-list> element

The <resources-list> node is the root of resources file.

4.2. Resource

```
<!-- ELEMENT resource (use-calendar)* -->
<!-- ATTLIST resource
      id          CDATA          #REQUIRED
      type        CDATA          #IMPLIED -->
<!-- time table of a resource -->
<!-- ELEMENT use-calendar EMPTY -->
<!-- ATTLIST use-calendar
      idref       CDATA          #REQUIRED -->
```

Example 7 - Portion of the DTD for the <resource> element

The <resource> node describe a resource, identified by its "id" attribute. Usual resource "type" attribute are "worker", "computer"...

You can link a resource to its calendar with the <use-calendar> element. A calendar indicates when this resource is (not) available (see below for further description of calendar). If a resource is not linked to any calendar, the resource will be considered as available every day.

4.3. Calendar

```
<!-- ELEMENT calendar (label?, day-types, default-working,
      default-nonworking, (day|timeperiod)*, start-on?,
      stop-on?, calendar*) -->
<!-- ATTLIST calendar
      id          CDATA          #REQUIRED -->
<!-- YYYY/MM/DD -->
<!-- ELEMENT start-on (#PCDATA) -->
<!-- ELEMENT stop-on (#PCDATA) -->
<!-- day covers two types of days: day of week {mon,
      tue, wed, ...etc}, the national days (holidays which takes
      place each year) mm/dd. -->
<!-- ELEMENT day (#PCDATA) -->
<!-- ATTLIST day
      type        CDATA          #REQUIRED -->
<!-- from YYYY/MM/DD to YYYY/MM/DD -->
<!-- ELEMENT timeperiod EMPTY -->
<!-- ATTLIST timeperiod
      from        CDATA          #REQUIRED
      to          CDATA          #REQUIRED
      type        CDATA          #REQUIRED -->
<!-- ELEMENT default-working (#PCDATA) -->
<!-- ELEMENT default-nonworking (#PCDATA) -->
```

Example 8 - Portion of the DTD for the <calendar> element

The <calendar> node is done to define the resources availability. By default, every days are considered as worked. You can then add to a calendar some not worked days with the following nodes:

- <start-on> indicates that days before this date are not worked.
- <stop-on> indicates that days after this date are not worked.

- **<day>** indicates with the attribute "type", the type of the given day. "type"'s value is referred to the type of days declared in the node **<day-types>**. The given day can be a day of the week taken in {mon, tue, wed, thu, fri, sat, sun} or a date, in short format MM/DD for the national day (e.g. day of nonworking type which take place each year).
- **<timeperiod>** indicates that days between "from" date and "to" date are of type indicated by the attributes "type". You can specify period of working type or non working type.

```
<!ELEMENT day-types (day-type*)>
<!ELEMENT day-type (label?, interval*)>
<!ATTLIST day-type
    id          CDATA          #REQUIRED>
<!-- Interval of time for day-type -->
<!ELEMENT interval EMPTY>
<!ATTLIST interval
    start       CDATA          #REQUIRED
    end         CDATA          #REQUIRED>
```

Example 9 - Portion of the DTD for the **<day-types>** element

The **<day-types>** node defines the root of the description of all the type of days that the calendar could use to describe the resource unavailability.

The **<day-type>** node define a type of day. A type of day is defined by a label (e.g. the name of the day type) and an interval of working time. If a **<day-type>** node doesn't have a child node **<interval>**, that means that the type of day defined is a non working type. The attribute "start" and "end" of **<interval>** is a time in format HHMM.

For example, look at the following time table:

```
<calendar id="c_john">
<label>John's calendar </label>
  <day-types>
    <day-type id="work_day">
      <label>Usual working</label>
      <interval start="0800" end="1200"/>
      <interval start="1300" end="1700"/>
    </day-type>
    <day-type id="day_off">
      <label>Holiday</label>
    </day-type>
  </day-types>
  <default-working
idref="work_day"/>
  <default-nonworking
idref="day_off"/>
  <day type="day_off">"sat"</day>
  <day type="day_off">"sun"</day>
  <timeperiod to="2004-06-12" from="2004-06-05" type="day_off"/>
</calendar>
```

This table means that resources linked to this table won't work :

- every week on sunday and saturday
- the january 1 of every year
- from june 5 to june 12 of year 2004

This calendar means that a resource works from 8 a.m. to 12 a.m. and from 1 p.m. to 5 p.m..

5. Scheduling file

```
<!ELEMENT schedule (constraint-date*, (task|milestone))>
```

```
<!ATTLIST schedule
    id ID #REQUIRED">
<!ELEMENT task (constraint-date*, priority, status?,
    reports-list?)>
```

Example 10 - Portion of the DTD for the <schedule> element

The <schedule> node is the root of the results from the scheduling. Each node <task> and <milestone> are element already existing in the project description. This file contains all the informations generated during the scheduling. That's why <task> and <milestone> can only have as children elements <constraint-date>, <priority>, <reports-list> and <status>. The <constraint-date> are essentially of type "begin-at-date" and "end-at-date". The <status> is computed according to past and planned activities for this task. The <reports-list> contains <report> elements and describes the planned activities. (See [Section 6 "Activities file" - Chapter II](#) for more informations on activities description.). The attribute "id" is a reference to the project associated to this results.

6. Activities file

```
<!ELEMENT activities ((task|milestone)*) >
<!ATTLIST activities
    id ID #REQUIRED">
<!ELEMENT task (reports-list)>

<!ELEMENT reports-list (report)+>
<!ELEMENT report EMPTY>
<!ATTLIST report
    idref IDREF #REQUIRED
    from CDATA #REQUIRED
    to CDATA #REQUIRED
    usage CDATA #REQUIRED>
```

Example 11 - Portion of the DTD for the <activities> element

The <activities> node is the root of the registered activities (past and future) associated to the project as id the attribute "id".

A <report> node is a detailed description of an activity. The attribute "from" indicates the begin date of the activities and the attribute "to" indicates the end date. The attribute "idref" is a reference to the resource id which is linked to this report activity. The attribute "usage" describes the percentage of time the resource will spend on the task during this activity.

Chapter III

Converting, scheduling and rendering

As we said at the beginning, three functionalities exist for Projman. Also, there are three ways of using Projman. Here is the synopsis of the Projman command line possibilities:

```
usage:
  1) To convert data:
    projman --convert [--arg=value] <xml_input_file> <xml_output_file>

  2) To render diagrams:
    projman --diagram [--arg=value] <projman_input_file> <output_image>
  3) To schedule the project:
    projman --plan [--arg=value] <projman_input_file> <output_file>
  4) To generate XML documents for call for tenders:
    projman --xmldoc [--arg=value] <projman_input_file> <output_file>
```

This can first import data in Projman format (by using `--convert` option) and then schedule the project (by using `--plan`) and then generate the wanted diagrams (by using `--diagram` option). The specificity of Projman is that you can schedule a project a single time and then generate the diagrams from the scheduling results. Each result of this operation are stored in XML files.

Now, we will describe how works each of Projman functionalities and all the options available for each mode.

1. Converting

The different options available to convert file are the following ones:

```
--in-format / -i
    specifies the format of the input.
    - planner
    - pygantt
    - projman

--out-format / -o
    specifies the output format.
    - projman [default]
      Generate activities_description.0.xml,
      resources_description.0.xml and
      project_description.0.xml if --project,
      --resources and --activities
      options are omitted.
    - planner
    - pygantt

--project / -p
    specifies the dest file path for the description
    of the project

--resources / -r
    specifies the dest file path for the description
    of the resources

--activity / -a
    specifies the dest file path for the description
    of the activities
```

If `<xml_input_file>` contains more than one project description, the conversion will generate an output file for each project description.

2. Scheduling

You will find in this section some things that you should know about those different algorithms.

Projman provides two ways to schedule a project : the standard (old) algorithm or another algorithm using constraint programming. Each algorithm tries to reduce the end of the project.

The options available for the scheduling mode are:

```
--include-references
    input file will be modified to include references to
    produced schedules
```

2.1. Default scheduling

The default algorithm is the fastest of both and will surely find a solution for a scheduling, even if all constraints are not satisfied. However, this algorithm handles resources assignments and resources calendar.

2.2. Constraint scheduling

This algorithm will try to satisfy all the constraints described in the project file. If some constraints are unsatisfiable, it won't find any solution. You should then correct your project file to make it find a solution.

Here are some hypotheses made by this algorithm :

- a resource works on a single task per day.
- a task is performed by a single resource. This means that if a task has more than one used resource, it will choose only one of these resources to work on this task.
- a task which doesn't have some used resources specified (nor its ancestors) is not affected by the two previous points.

3. Rendering

TO VALIDATE NOT SURE!!!!!! Projman is able to draw the Gantt diagram in the following format : HTML, SVG, ARG, BMP, CUR, DCX, EPS, FLI, FPX, GBR, GIF, HTML, ICO, IM, IPTC, JPEG, MIC, MPEG, MSP, PCD, PCX, PDF, PNG, PPM, PSD, SGI, SUN, SVG, TGA, TIFF, WMF, XBM and XPM .

The diagram should look similar in all formats.

The different options available are:

```
--diagram-type
    specifies the type of diagram to generate
    this option is required
    - resources (describes resources activities
    in past and future)
    - gantt
--selected-resource
    specifies the id of the resource to take
```

```

in account for resources diagrams
[default=all resources]

--renderer
specifies the output rendering for resources
diagram [default=png] use only with schedule input
- html (bugs)
- png

--timestep
timeline increment in days for diagram
[default=1]

--depth
specifies the depth to visualise for diagrams
[default=0, this means all the tree]

--view-begin
begin date for diagram view [default=date begin project]
ex: yyyy/mm/dd

--view-end
end date for diagram view [default=date end project]
ex: yyyy/mm/dd

```

4. XML documents

This mode of action proposes tree view to answer to call for tenders. Each view is an XML file which can be included in a global answer to call for tenders.

The different options available are:

```

--view
Specify the type of view generated
- [tasks-costs]: Will create a table in which the project's tasks
are listed with all the main informations (duration, cost, resources)
- tasks-list: Will dump tasks information.
- tasks-dates: Will dump begin, end date and global cost for
tasks.

--format
set output format.
- [docbook]: respectful of format docbook 4.2.
- html: NOT IMPLEMENTED YET
- csv: plain text with separator NOT IMPLEMENTED YET

```

Appendix A

Annexes

1. Related programs

If you're looking for GPL'd project management software, you should try [ToutDoux](http://toutdoux.sourceforge.net/) [http://toutdoux.sourceforge.net/], that aims to be a complete tool and needs Gnome libraries or [Mr Project](http://mrproject.codefactory.se/) [http://mrproject.codefactory.se/], a M\$ Project clone (needs Gnome libraries too). Another possibility is [TUTOS](http://tutos.sf.net) [http://tutos.sf.net], written in PHP and which provide a bunch of others functionalities like a bug tracking system, an address book...

If you only want to draw Gantt diagram, try [QtGantt](http://www.gumbley.demon.co.uk/qtgantt.html) [http://www.gumbley.demon.co.uk/qtgantt.html], that needs Qt or [JGantt](http://www.egantt.com/jgant/) [http://www.egantt.com/jgant/], a Java component.