

TaskJuggler

The Best Laid Plans

Enterprises and service providers are not the only ones to rely on computer supported task and project planning. Clubs and non-profit organizations can save a lot of time, money, and headaches with the right approach. TaskJuggler helps you organize your resources and provides a clear overview.

BY FREDERIK BIJLSMA

AND PATRICIA JUNG



Tasks and projects that need organizing have accompanied mankind down through the centuries. Unfortunately, people and more recently programs, that can assign resources and tasks in a meaningful and above all efficient manner have always been in short supply. *TaskJuggler* [1] by Chris Schläger and Klaas Freitag not only provides this kind of functionality, but turns out to be a genuine all-round talent that can optimize any given scenario that can be defined in terms of projects and resources.

The program allows you to define cost centers and simple controlling facilities, along with shifts and priorities. The pro-

gram accepts input in the form of text files with the extension `.tjp` that describe objects. The following syntax tells the program to parse a file

```
taskjuggler filename.tjp
```

TaskJuggler produces reports in *HTML* or *CSV* file format that are perfectly suited for Internet publications, or for additional manipulation within spreadsheets like OpenOffice. As an alternative, the program can also produce XML data, that can then be converted into any other required target format, allowing easy migration to your content management system.

Advanced Functionality

Let's take a look at the developer version 1.9.2, which provides quite a few new features such as the CSV Reports mentioned previously. In addition to a C++ compiler, such as g++ from the "GNU Compiler Collection", GCC, you will also require the X Window system developer package. Despite what the documentation on the Taskjuggler website, <http://www.taskjuggler.org/>, maintains, you will also need at least version 3.1 of the Qt library.

To unpack the TaskJuggler package, enter the following

```
tar xjf taskjuggler-1.9.2_?
```

GLOSSARY

CSV: Files in the "Comma Separated Value" format simplify the exchange of data between spreadsheets. The cell contents are separated by commas and newlines, although any formatting will be lost on conversion.

C++: The C++ programming language was developed more than 20 years ago to bring data abstractions, object-oriented programming, and other modern concepts to the C programming language. Since then, C++ has achieved ISO certification and is available on almost any architecture or operating system.

Docbook: A "Document Type Definition", that is, a description of the tags that can be used in an XML file. Docbook defines the elements that a book comprises. This allows text mark-ups that can be converted to professional printed or online formats. Open Source projects use the Docbook format for at least part of their documentation.

CPAN: The "Comprehensive Perl Archive Network" at <http://cpan.perl.org/> provides software, modules and documentation for Perl.

Gantt diagrams: Show the timeline for tasks that need to be finalized to allow a project to be completed. As you can see from Figure 1, each task occupies a single line in the diagram. Gantt charts are particularly useful for projects where the number of tasks that is to be planned is not too large. They owe their name to the American, Henry Laurence Gantt (1861-1919), an engineer and management consultant. Gantt charts were used successfully in Arizona in the 1930s for the Hoover Dam building project.

```
unstable.tar.bz2
cd taskjuggler-1.9.2_unstable
```

Then launch the compilation process by entering

```
./configure --with-kde-support
make
```

You can omit the `--with-kde-support` option, if you do not have KDE. The configuration routine is designed to provide as much functionality as the host machine's environment can support. As an example, it re-generates the documentation files if it can locate the **Docbook** libraries.

If Perl and the `XML::Parser`, `Postscript::Simple`, `Date::Calc`, `Class::Method Maker`, and `Data::Dumper` CPAN modules are installed, the build process will also create a program called `tx2gantt`, which creates **Gantt diagrams** as shown in Figure 1.

After completing the `make` process, go on to assume system privileges before installing TaskJuggler by typing `make install`.

Listing 1: Shift definitions

```
01 project shifts "Duty Schedule
   SysAdmin Team" "$Id" 2002-06-
   01 2002-08-01 {
02     dailyworkinghours 8
03     yearlyworkingdays 256
04 }
05 flags hidden
06 shift phonesupport "Phone
   support" {
07     workinghours mon 9:00 -
   12:00
08     workinghours tue 9:00 -
   12:00
09     workinghours wed off
10     workinghours thu 14:00 -
   17:00
11     workinghours fri 9:00 -
   12:00
12 }
13 shift studenthours "Student
   Hours" {
14     workinghours mon 9:00 -
   14:00
15 [...]
16     workinghours fri 9:00 -
   14:00
17 }
```

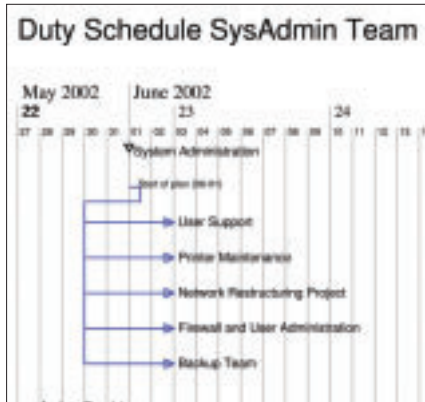


Figure 1: A Gantt chart for a project

Projects, Shifts, & Resources

It makes sense to create a directory for your own projects, to give TaskJuggler somewhere to store your definitions and reports. The TaskJuggler website provides a few examples [2] to help explain how to use the application.

We can take a look at the Examples directory in the TaskJuggler source download. This contains a `ShiftSchedule.tjp` and details the four components of the project: the definition, the levels, the available resources, and the tasks.

Let's take a look at the actual *project* section of this example first (see Listing 1). *shifts* provides an identifier, which is followed by the project name – this project deals with shift work for the system administration team.

The last few parameters indicate the timeline for the project in YYYY-MM-dd format. You should allow some leeway

Listing 2: Resource definitions

```
01 resource joe "Joe Bughunter" {
02     vacation 2002-06-10 -
   2002-06-13
03 }
04 resource khaled "Khaled Safri"
   {
05     shift studenthours
06 }
07 [...]
08 resource anders "Anders
   Gundstrom" {
09     maxeffort 0.8
10 }
11 resource paul "Paul Gutier" {
12     vacation 2002-07-02 -
   2002-07-08
13 }
```

when defining the end date, as tasks outside of the project timeline are simply ignored.

The third parameter you need is a version number. This can be a simple "1.0"; if you use a version control system to manage your TaskJuggler files, you can specify a wildcard ("*\$Id" for CVS).

These mandatory parameters can be followed by optional details in brackets, such as the daily working hours (eight hours in our example) and the number of workdays per annum (256 in the example).

The *shifts* object can now be used to define periods of shift-work. Shifts are used to accumulate the time accounts of the people working at the same time. The *workinghours* object defines the specific daily work periods. Days are abbreviated to *tue* for Tuesday, *wed* for Wednesday etc.). The keyword *off* indicates a day without working hours within a project period. If you miss a day, TaskJuggler will assume the default working hours of the project for this shift.

Each shift is assigned an ID (for example *phonesupport* for the time that the members of the sysadmin teams will spend on the receiving end of the hotline) and a title.

Listing 3: Task definitions

```
01 task sysadmin "System
   Administration" {
02
03 # The following is not really
   a task but simply
04 # defines the project starting
   date.
05 task start "Start of plan" {
06     start 2002-06-01
07     milestone
08     flags hidden
09 }
10
11 task usersup "User Support" {
12     depends !start
13     duration 2m
14     shift phonesupport
15     priority 900
16     allocate joe { alternative
   anders, khaled, sally select
   minloaded }
17 }
18 [...]
19 } # End of Sysadmin Tasks
```

The staff members affected by this project are indicated by the use of the *resource* keyword, as shown in Listing 2. In addition to a unique descriptor (such as *joe*), the full name is useful as a description. The *vacation* attribute can optionally be used to define a vacation period; *maxeffort* defines a factor that allows you to calculate part-time work. Instead of a forty-hour working week, Anders Gundstrom only works $8 \times 0.8 = 6.4$ hours. The *shift* keyword binds Khaled Safri to special working hours; in this case, the times defined for students in Listing 1.

Finally, the most important thing: the tasks to be completed (see Listing 3). Our example first defines a *task*, *sysadmin*, which is then split up into individual jobs in curly brackets. The *start* point defines a *milestone*, at which other tasks commence. User support *depends* on completion of the *start* job, and cannot occur prior to June 1 2002. The milestone will not appear in reports due to the *flags hidden* tag.

We can now assign the support task, which we will be planning for the next two months (*duration 2m*) to the *phone-support* shift with a priority of 900. 1 indicates something extremely unimportant, while 1000 has utmost priority. The *allocate* attribute assigns staff to the task. In this case this means selecting those members of staff from a group comprising *joe*, *anders*, *khaled*, and *sally* who have the lightest workload from other tasks (*select minloaded*). The *maxloaded* selection criterion is the opposite of this,

Listing 4: Sallys Assignment Schedule

```
htmlweeklycalendar "Calendar-
sally.html" {
  headline "Assignment Schedule
for Sally"
  columns schedule
  hidetask 1
  hideresource
  ~isresource(sally)
}
```

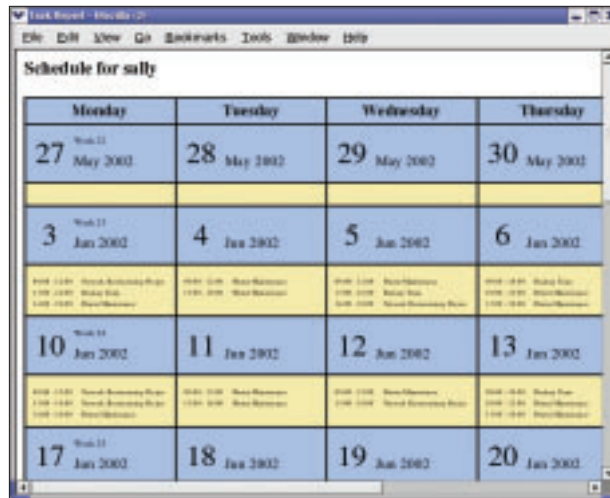


Figure 2: HTML Calendar for Sally

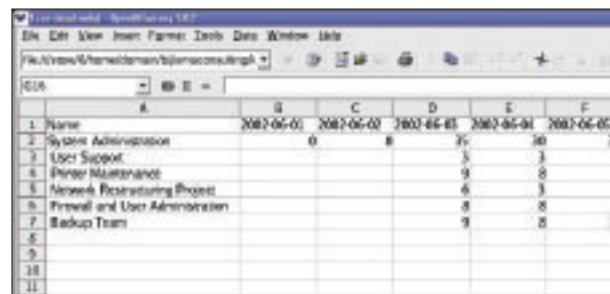


Figure 3: OpenOffice with the workload data from TaskJuggler

and selects the member of staff with the heaviest workload. *Order* selects the first member of staff without an assignment, and *random* simply picks anyone.

Simple Planning

As soon as the scope of the project has been specified, TaskJuggler can provide planning support: Adding the following line to the *.tj* file

```
xmlreport "ShiftSchedule.tjx"
```

tells TaskJuggler to create an XML report and store it in a file called *ShiftSchedule.tjx*. Of course you can view this file with an XML editor, or similar tool, but its

Listing 5: Overview of the number of working hours per task and day

```
csvtaskreport "effort.csv" {
  columns name, daily, effort
  start 2002-06-01
  end 2002-07-01
  hidetask hidden
  loadunit hours
}
```

true value is revealed when you need to import or export data. The appropriate DTD is available from http://www.taskjuggler.org/show_dtd.php. You can then feed the XML file to the Perl script mentioned previously, *tjx2ganttt* [3], to create a Gantt diagram.

In addition, TaskJuggler is perfectly capable of generating useful HTML reports without any additional tools. Adding the code from Listing 4 *htmlweeklycalendar* to the *.tj* file for Sally's calendar in Figure 2 will create a week-by-week plan of the project period in *Calendar-sally.html*.

The *isresource(sally)* function filters the assignments for the resource *sally*, and *hideresource* hides any resources that do not (~) match this criterion. *columns schedule* outputs a detailed schedule using this data. Leaving out the *hidetask 1* line, inserts a line for notes between the date and task in the HTML

calendar.

CSV reports are also defined in the *.tj* file, as Listing 5 shows. *csvtaskreport* outputs the individual names and *effort* in hours (*loadunit hours*) for all non-hidden tasks for the period between 06.01.2002 and 07.01.2002 (see Figure 3).

Good planning takes you half-way home

No matter if you need *who-will-be-working-for-how-long-on-what?* reports, or simple assignment plans for your staff – TaskJuggler offers a range of functions that could easily fill a reference manual. The *Examples* directory in the *taskjuggler* source archive provides more examples. This directory and the sample files also offer a number of tips, for example, on the use of macro scripts, and multipart projects. ■

INFO

- [1] TaskJuggler: <http://www.taskjuggler.org/>
- [2] TaskJuggler projects: <http://www.taskjuggler.org/example.php>
- [3] *tjx2ganttt* Perl script: Installed by default with TaskJuggler