frePPLe

A free Production Planning Library

MANUAL
VERSION 0.6.0
JANUARY 2009

* * *

Typeset by Wikipublisher

CONTENTS

CHAPTER

1 Introduction

FrePPLe aims at building a lightweight open source framework for modeling and solving production planning problems.

Production planning software traditionally has been an area with plenty of home-grown, extremely specialised and/or very primitive solutions.
Strangely enough, while creative and innovative open source solutions pop up in all computing areas, production planning software still tends to be a very closed world full of academic, proprietary and expensive solutions. Till now…
Frepple is the first open source production planning toolkit for your day-to-day planning problems.

For the developer community, the project is also trying to establish a common ground framework for planning applications. Rather than rebuilding the basic foundation from scratch over and over again, developers can now leverage a proven framework to extend with their own extension modules.
New workflows and functionality can now be built much quicker and easier.

The word "free" in the project name refers to liberty, not price. Think of "freedom of speech" rather than "free beer": see *the free software definition*[1].

Features

FrePPLe has two main components.

[1] www.gnu.org/philosophy/free-sw.html

1. The first one is a **core library** containing the model and the solving algorithms. It is generic and can be used in a number of applications.
2. A second component is a flexible **user interface and database layer** to support the core library. It takes care of the maintenance of input data, reporting of the plan results, and data integration to other systems.

The key features of each component are:

1. **FrePPLe core library**
   - FrePPLe comes as a **'library' developed in C++**.
     It has no graphical user interface and requires to be deployed as part of another application.
     Different applications are envisioned:
     - Standalone application for use on the command line
     - Accessable from programming languages such as Java, Python, Perl or Visual Basic.
       The interface to Python is exceptionally rich and allows direct interaction with the objects.
     - Can be linked into your own C or C++ application
   - **Modeling and solving framework for discrete manufacturing environments.**
     Key modeling constructs are:
     - Item
     - Buffer
     - Resource
     - Operation
     - Demand
   - **Heuristic "MRP-like" solving algorithm** respecting capacity, material and leadtime constraints.
   - **XML**-based data input and output, in addition to the public C++ API.
   - Very **fast**!
     Performance and scalability have been a consideration from day one…
   - **Extensible and customizable architecture.**
     New modeling constructs and solving algorithms can be developed in C++ and loaded as a plugin module.
   - Embeds **Python** as scripting language.
     The embedded interpreter has access to the frePPLe objects in memory, combined with the rich functionality of the Python libraries. The powerful combination allows flexible and performant scripting, integration and customization.
   - Supported on **linux and Windows** environments.
   - Licenced under the **GNU lesser general public license**.
2. **FrePPLe user interface and database layer**
   A planning solution consist of much more than the core solver algorithms…
   It includes data maintenance, reporting, data integration to other systems, workflows, job schedules, etc…
   A front-end for the core library is required to meet these requirements with a maximum of flexibility.
   - Based on the **Django web application framework**.
     Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design.

For frePPLe it provides an excellent toolkit:
  - Flexible and portable modeling of database layer: powerful and intuitive object-relational mapper
  - Flexible and performant framework for web applications: auto-generated administration user interface, template system, cache system, internationalization, ...
  - Performant and scalable
- **Highly customizable** and extendible.
- The web application infrastructure can be **deployed on a web server, supported by a backend database**.
  It can also be installed as a **standalone application** on the user's PC.
- Supports the **PostgreSQL, MySQL, SQLite and Oracle** databases.
- Full support for **internationalization** and **localization**.
  The user interface supports unicode, which allows characters of any language to be handled.
  Reports can be translated to the user's local language.
- Supported on **linux and Windows** environments.
- Licenced under the **GNU lesser general public license**.

Architecture

The frePPLe components can be used in a number of different ways.

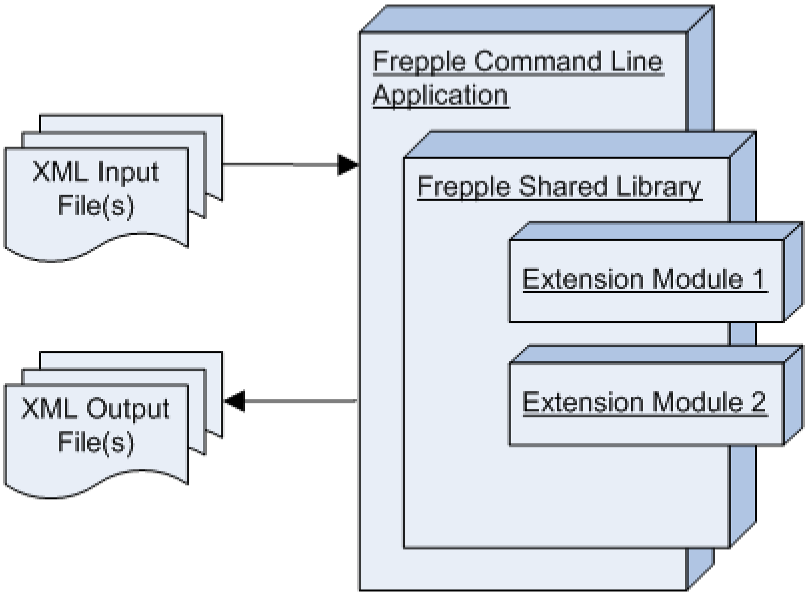
1. Core library
2. User interface and database layer

Core library

The frePPLe binaries are a collection of shared libraries: a core library frepple.so (frepple.dll on Windows) and an additional shared library for each extension module.
The extension modules are loaded dynamically as plugins by Frepple.

The frePPLe shared library can be used in different ways by applications.
Below is a list of some common ways to deploy frePPLe, but additional scenarios are definately feasible.
The main development efforts are currently focussed on the first and the last two scenarios.

Command line application



A simple command-line application is available.

The application reads a set of XML files or from the standard input.
It executes all commands defined in the input data (which will typically also involve some python code to solve the model and write the results back into flat files or a database) and then exits.
The program exit code reflects any processing errors.

Example usage:

```
frepple file1.xml
frepple file2.xml file3.xml
frepple dir_with_xml_files
command | frepple
```

Use the option "-help" or "-?" to get a list of possible flags that can be passed on the command line.

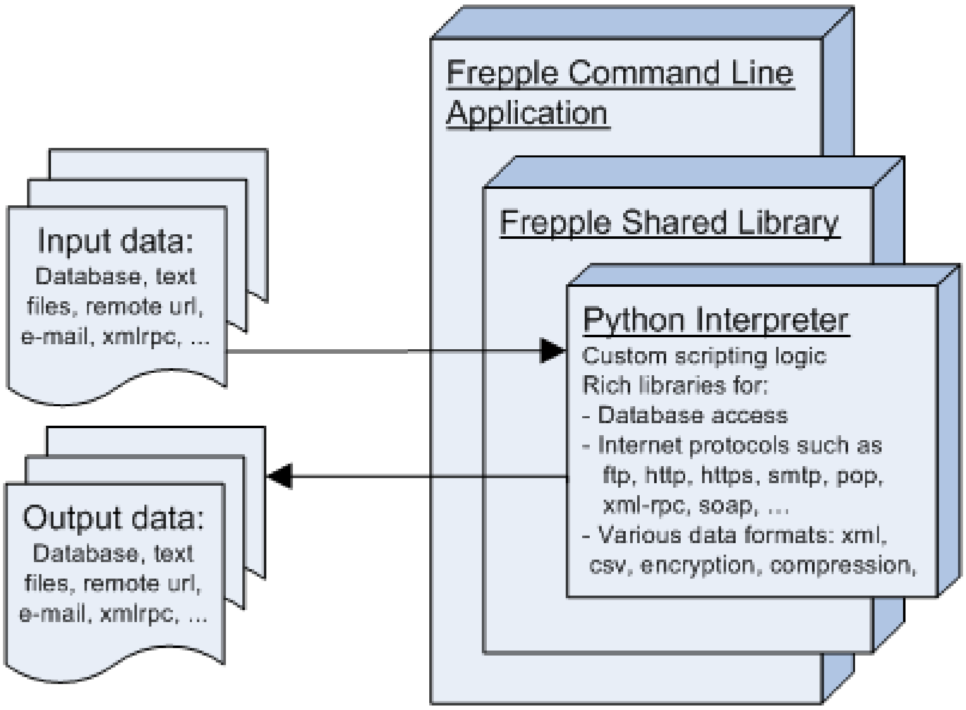
This command line application is used for all test cases.

Command line application with Python scripting

In the previous setup the XML input and output files are supplied externally.

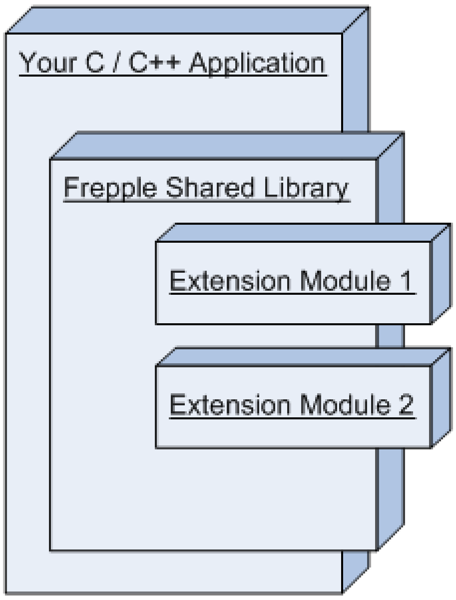
FrePPLe comes with an embedded interpreter for the *Python*[2] language.
Python is a dynamic object-oriented programming language. It comes with extensive standard libraries for database access, a wide range of internet protocols (such as ftp, http, https, smtp, pop, xml-rpc, soap, . . .), various data formats (such as xml, csv, compression, encryption), . . .
The Python interpreter has a rich API to access the frePPle objects in memory. This allows custom logic to be implemented in an easy and flexible way, with full access to the rich Python standard libraries.

[2] www.python.org



For a majority of applications this will be the recommended setup.

Your C or C++ application links with frePPLe



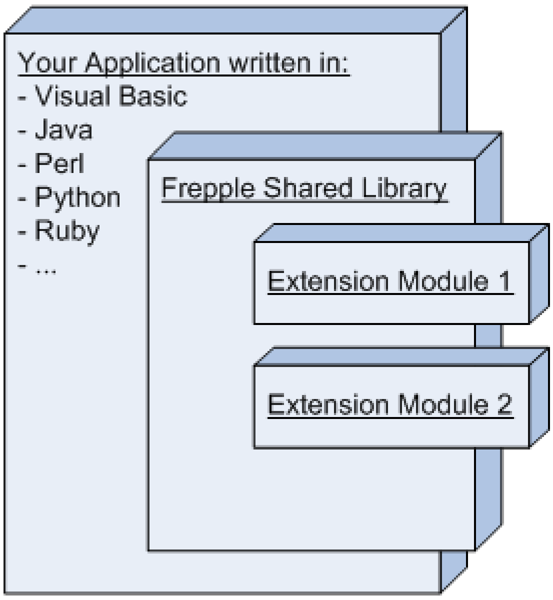
Your application can be link with the frePPLe shared library.

Use the header file plannerinterface.h for the high-level interface declarations.
Use header file frepple.h when you need low-level access.

Since frePPLe is coded in C++:

- C applications will need some wrapper code to catch exceptions correctly and assure C linkage.
- Because of the C++ name mangling frePPLe and your application will need to be compiled by the same compiler.

Your java/perl/ruby/VB/.NET application accesses the frePPLe shared library



Most modern languages and tools have the capability to access functions in shared libraries.

SWIG[3] is a tool that can help to generate the integration code with a wide range of high-level languages, such as Java, Ruby, Perl, Tcl, PHP, . . .
An example setup is provided in the subdirectory *contrib/scripting*.

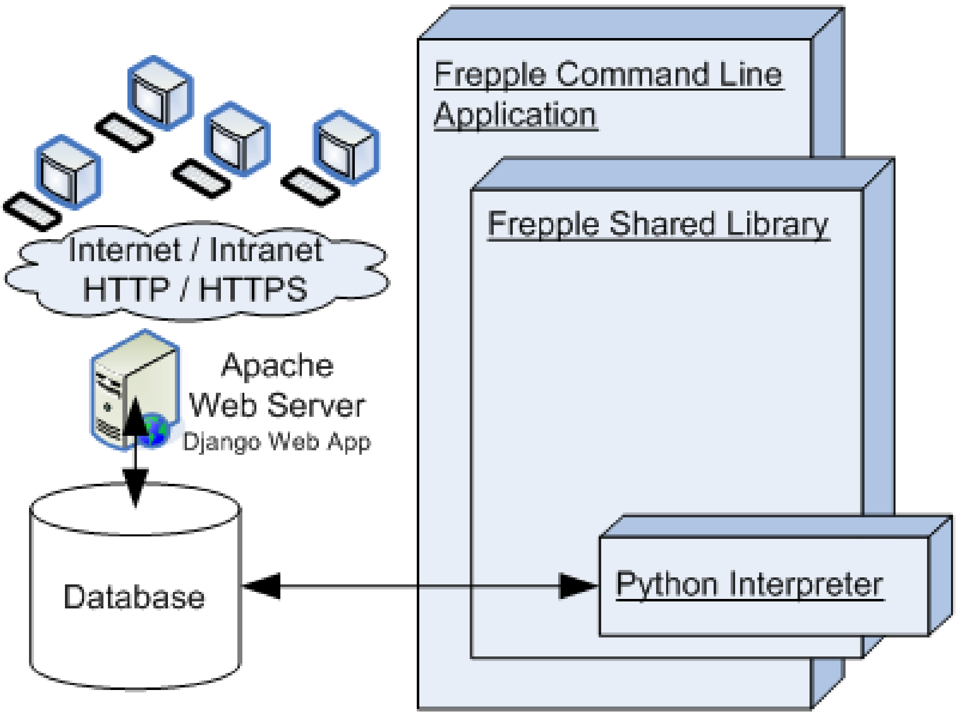
When building applications in this way, remember that the scripting language will load the frePPLe shared library and all memory allocated by frePPLe (which can be quite a lot!) will be owned by the scripting language process. For large models this is not be a very appropriate integration method.

Django frontend for frePPLe

Django[4] is an impressive web application framework written in the Python language.
It allows quick and easy definition of the data model, automatically creates a administration user interface and allows you to construct performant and scalable web sites.

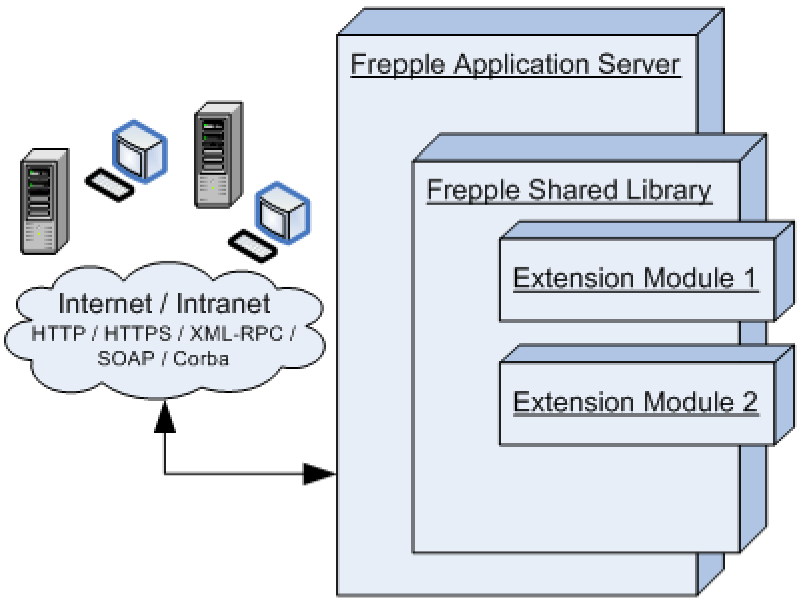
FrePPLe then reads from and writes into this Django database.

[3] www.swig.org
[4] www.djangoproject.com



The subdirectory *contrib/django* provides a reference Django model for frePPLe.
In a real-life implementation you will typically develop your own data model. You'll build web pages to support the user's workflows, and then write the appropriate mapping between your data model and the frePPLe internal data structures.

FrePPLe as a web service



FrePPLe comes with an extension module that implements a SOAP web service.

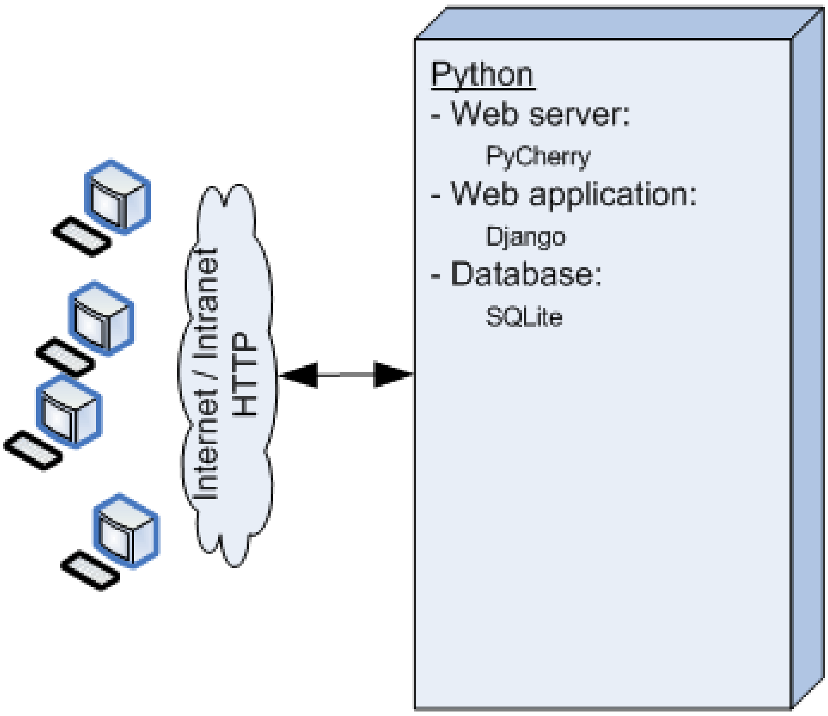
In a Service Oriented Architecture, frePPLe will hold the plan information in memory and make it available on-line. Other systems can use the service to query and update the information to build composite applications.
Users can also directly access also the information from e.g. Excel (using the office Web Service Toolkit).

User interface and database layer

A planning solution consist of much more than the core solver algorithms...
It includes data maintenance, reporting, data integration to other systems, workflows, job schedules, etc.
A front-end for the core library is required to meet these requirements with a maximum of flexibility.

FrePPLe includes a user interface based on the Django web application framework. The user interface can be deployed in different architectures, depending on the requirements. With increasing levels of scalability and performance, we can basically distinguish the following three main scenario's.

Standalone/all-in-one application



The windows installer includes a standalone application.
The application is an all-in-one installation containing:

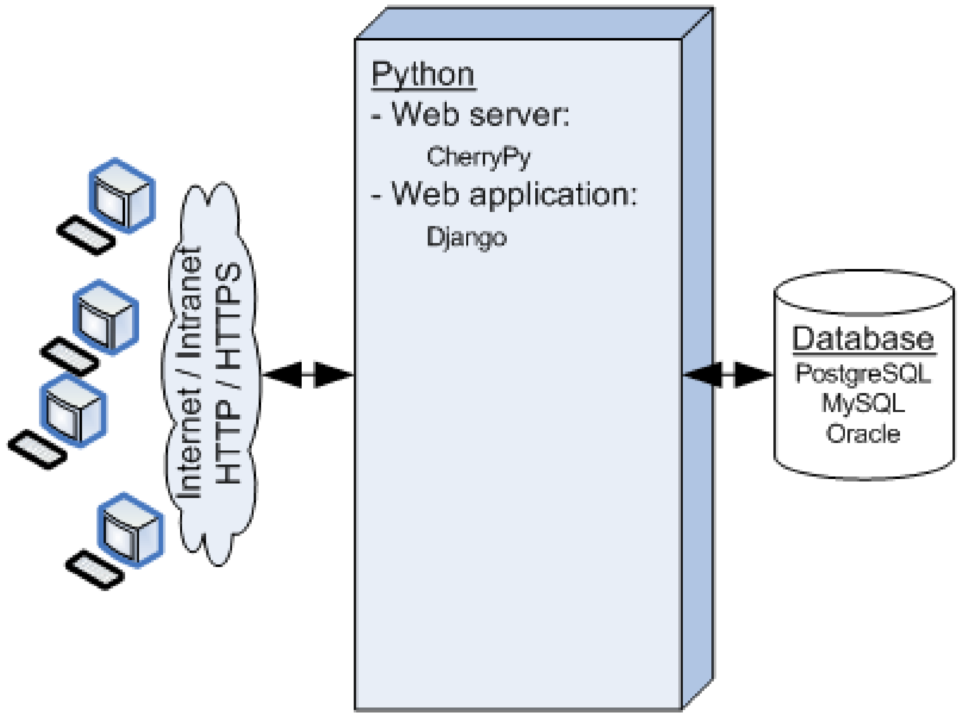
- Python interpreter and Python libraries.
- Web server CherryPy, written in python.
- Django web application.

- Database SQLite, which is part of the Python standard library.

This one-stop installation package (< 10MB download) makes it very easy to get started with frePPLe, as a tutorial or for educational purposes. It is also suitable to deploy frePPLe as an application to a user's PC.

This configuration can only be recommended for single-user access to small models.

Python application and a database



The SQLite database does an excellent job for relatively small datasets. But for the complex reporting queries used by frePPLe it is no match for the "real" database applications.
As a first measure for increasing scalability and performance of the application, the database needs to be separated out. FrePPLe supports the Oracle, MySQL and postgreSQL databases.

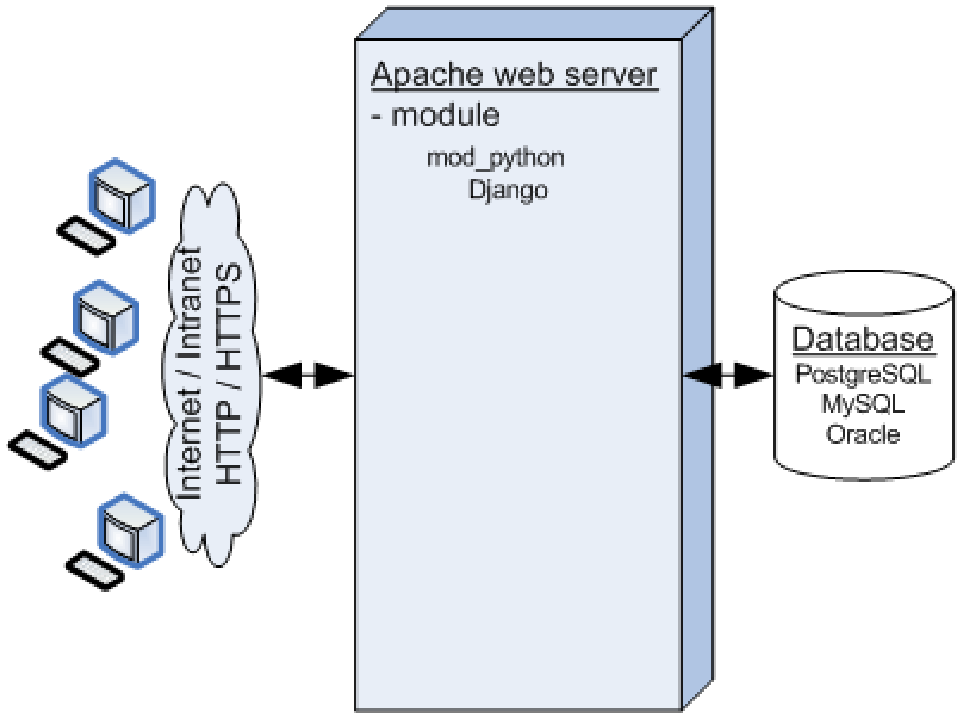
With this configuration a few users can simultanenously access frePPle.

Apache web server with mod_python and a database

This is the preferred deployment option for production servers!

Apache is now used as the web server. With the mod_python module it executes the Django python code.
The Apache server assures excellent scalability, performance and security.

Medium-volume sites will typically have a single Apache web server and a single database server.
High-volume sites with plenty of concurrent users can deploy additional components to guarantee



the right scalability and availability of the system: memory caches, separated web servers for static and dynamic content, replicated databases, load balanced web servers, enterprise authentication such as LDAP, . . .

CHAPTER

2

Download and install

The frePPLe project lives on the *www.sourceforge.net*[1] open source software development web site, where all release files and the source code are hosted.

Here is a link to the download page http://sourceforge.net/project/showfiles.php?group_id=166214

The project distributes the following formats:

- **Windows installer** (32-bit)
- **Source code tar-file** for all platforms
- A **VMware virtual machine** with a fully configured demo environment on Linux
- Access to the **Subversion source code repository** for the latest developments

2.1 Installing on Windows

[1] sourceforge.net/projects/frepple

2.1.1 Windows installer

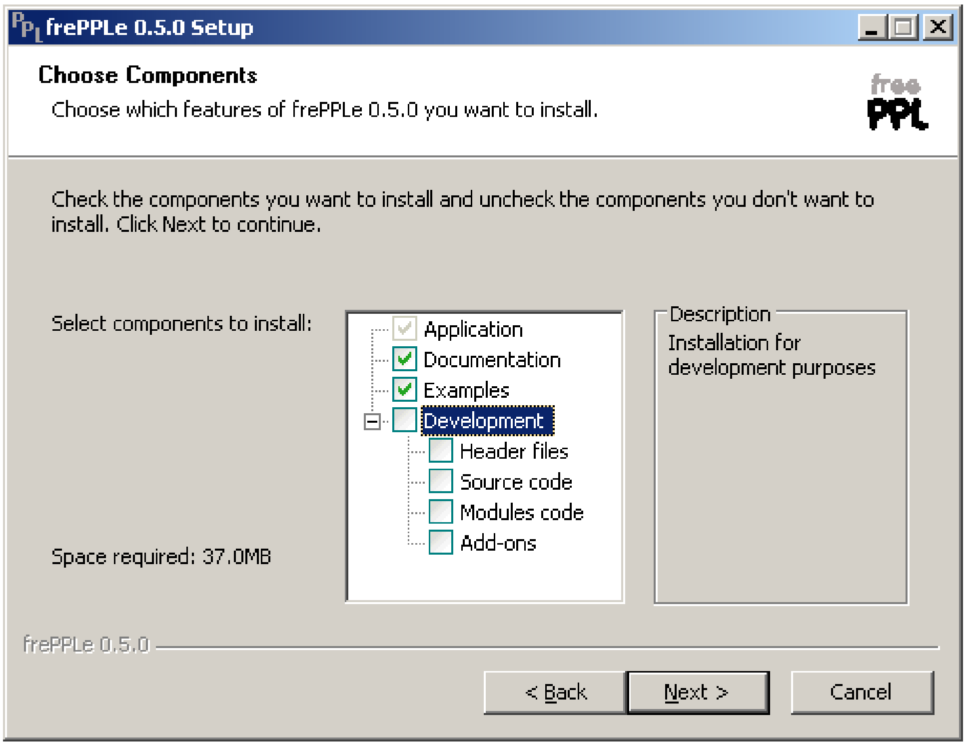
Installing and uninstalling frePPLe is straightforward, and follows the normal Windows conventions.

After accepting the licence agreement, the installer will guide you to select:

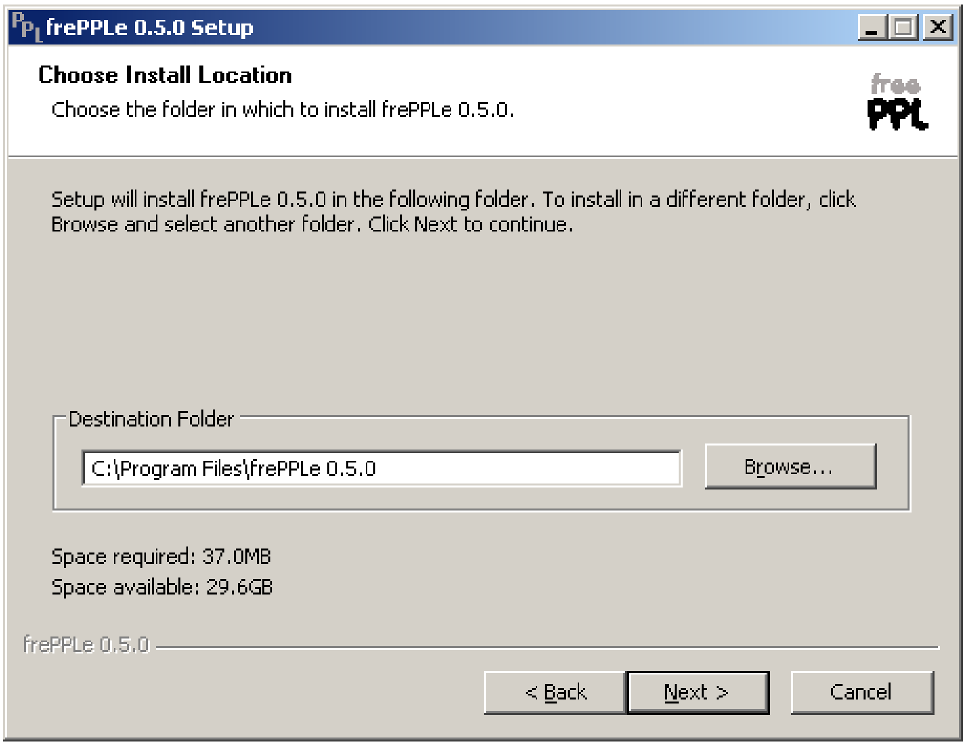
- The components to install
- The installation directory
- The database connection parameters

With all options included the installation requires less than 25 MB of disk space.

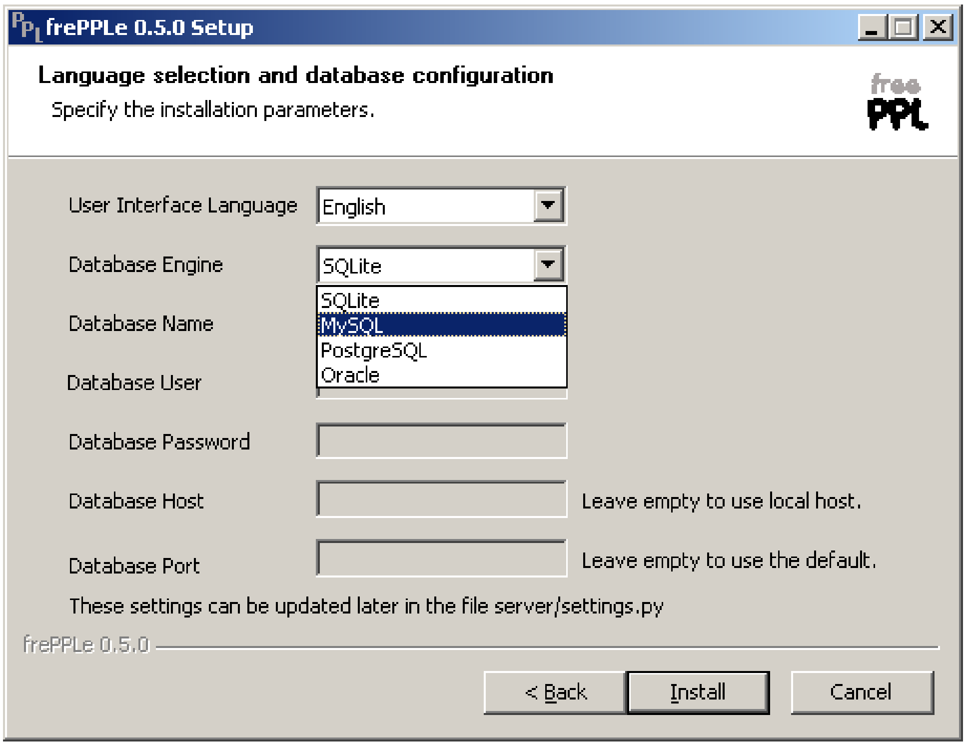
1. **Select the components to install.**



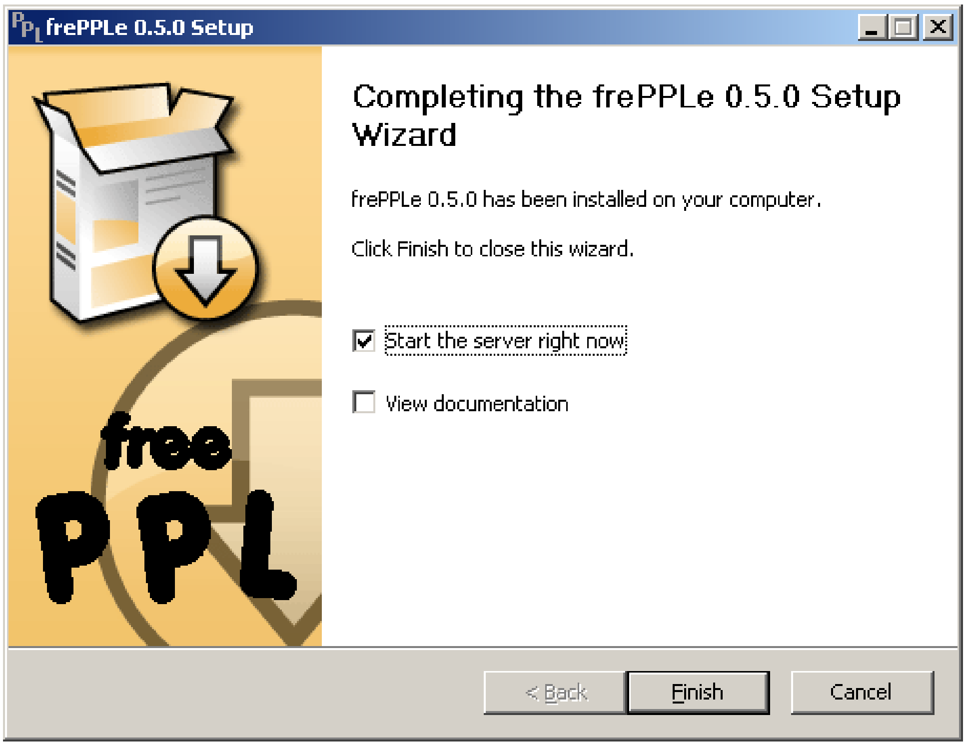
2. **Select the installation directory.**



3. **Select the installation parameters.**
   Two types of parameters are specified during the installation:
   3.1. language for the user interface
   3.2. database connection parameters
   FrePPLe supports the MysSQL, PostgreSQL, SQLite and Oracle databases. The installer will detect which of those you have installed on you computer and allow you to choose one. The SQLite database is included with frePPLe, allowing you to get started very quickly.
   For MySQL, Oracle and PostgreSQL you need to specify the database name, the database user and its password, and the host and port number of the database engine. The database and the database user have to be created by the database administrator. The frePPLe database tables will be created when you first start the server.
   For SQLite you only need to specify the database name. You selections are saved in the file server/settings.py. The file can later be edited with a text editor when required.



4. **Finish**
   At the end of the installation you can choose to start the server immediately. Point your browser to the url, and you're up and running!



The installer provides:

- Command line application
- Server application which bundles a python interpreter, python libraries, web server, django web application and database
- Documentation
- Development libraries
- Source code

It is possible to have multiple installations in parallel on the same computer. They need to be installed in different directories, and you need to set the environment variable FREPPLE_HOME to point to the directory with the version you want to run.

2.1.2 Compiling under windows

Different options exist to compile Frepple under windows:

- Microsoft Visual C++ Compiler (p 15)
- Cygwin Compiler (p 15)

Note that executables and extension modules created by these compilers are not compatible with each other.

Compiling using Microsoft Visual C++ compiler

Frepple comes with Microsoft Visual C++ projects and workspaces to build Frepple.
The solution file is **contrib/vc/frepple.sln** and more detailed build instructions are provided in the README.txt file in this directory.

The project configuration files are generated with version 9 of Visual C++ and (in the Microsoft tradition) are not compatible with earlier releases. :-(
A free version of the compiler and the IDE, called "Visual C++ 2008 Express Edition", can be downloaded from the Microsoft website.

You will also need to install:

- Python 2.5.x
- Xerces-c 2.8

The include and library directories of these tools need to configured in Visual C++ development environment: navigate to *tools* > *options* > *VC++ directories* to do this.

Compiling using the Cygwin compiler

Cygwin is a Linux-like environment for Windows. The Cygwin environment can be downloaded free of charge from the *Cygwin website*[2].

The build instructions on Cygwin are identical to the Linux and Unix platforms.

Compared to the other platforms and compilers, the Cygwin executables are considerably slower.
Consider the Cygwin build as a test and development setup for a *nix environment.

[2] www.cygwin.com

2.2 Installing on Linux, Unix and Cygwin

1. Build instructions
2. Compiling from the Subversion repository
3. VMware virtual machine

2.2.1 Build instructions

The following describes the steps you need to build frePPLe.

1. Update your system with the development software packages.
    - **gcc**, v3.4 or higher
      Front end for the GNU compiler suite.
    - **gcc-c++**, compatible with gcc release
      GNU C++ compiler.
    - **xerces-c**, v2.7 or 2.8
      Xerces is a validating XML parser provided by the Apache Foundation.
      You need to install the libraries as well as the development libraries.
    - **python** v2.4 or v2.5
      Python is a modern, easy to learn interpreted programming language.
      See http://www.python.org for more information. The language is used to a) run the test suite, b) script custom logic in frePPLe and c) to run the web application framework Django.
      You need to install the language as well as the development libraries.
2. Change to the installation directory.
3. Issue the command '**./configure**' to specify the build options and detect the specifics of your platform.
   Use the command './configure —help' to see the list of available options.
4. Issue the command '**make all**' to compile the code.
5. Optionally, issue the command '**make check**' to run the test suite.
   Not all tests are currently passing, so you shouldn't be worried about a couple of failures. :-)
6. Issue the command '**make install**' to install the files.
7. You can issue the command '**make clean**' to free the disk space used during the build and test phases.
8. Optionally, if you are interested in some of the add-ons in the contrib subdirectory, follow the instructions in the README.txt file in each of the add-on directory.
   You may need to install additional software components for a certain add-on. As a reference, here is a brief summary list of those components:
    - **Django**, v1.0.2
      A web application framework written in Python.
      FrePPLe supports PostgreSQL, MySQL, Oracle and SQLite as the database.
      In addition Django needs the python database driver for your database, the *apache web server*[3] and *mod_python*[4].
      Visit the *django website*[5] for full details.
      Later Django versions may or may not work with frePPLe...

[3] httpd.apache.org
[4] www.modpython.org
[5] www.djangoproject.com

- **SWIG**, any version should do
  SWIG is a software development tool that connects programs written in C and C++ with a variety of high-level programming languages. SWIG is used with different types of languages including common scripting languages such as Perl, PHP, Python, Tcl, Ruby and PHP.
- **GLPK**, any version should do
  The GLPK (GNU Linear Programming Kit) package is intended for solving large-scale linear programming (LP), mixed integer programming (MIP) and other related problems. It can be downloaded from http://www.gnu.org/software/glpk/glpk.html
- **NSIS**, version greater or equal to 2.07
  NSIS, which stands for "Nullsoft Scriptable Installation System", is a free scriptable win32 installer/uninstaller system that doesn't suck and isn't huge.
  This program can be downloaded from http://nsis.sf.net and you'll only need it if you are planning to create a windows installation package.
- **py2exe**, 0.6.6 or later
  Py2exe is a Python Distutils extension which converts Python scripts into executable Windows programs, able to run without requiring a Python installation.
  The software is used only when creating the windows installer. It can be downloaded from http://www.py2exe.org/.
- **CherryPy**, version 3.0.1 or later
  CherryPy is a pythonic, object-oriented HTTP framework.
  FrePPLe uses it as web server embedded in the Windows stand-alone application. The software is used only when creating the windows installer. It can be downloaded from http://www.cherrypy.org/.

9. Optionally, you can use the FREPPLE_HOME environment variable to point to your installation directory.
   See the section on environment variables for other environment variables that influence frePPLe and may need updating.

2.2.2 Compiling from the Subversion repository

To work with the code from the repository, follow the steps below.
Step 3 is the main difference with the build process from a distribution.

1. Your machine will need the following software components **in addition** to the ones listed for compiling from a distribution file:
   - **autoconf**, v2.59 or later
     Gnu Autoconf produces shell scripts to automatically configure software source code packages. This makes the source code easier to port across the different *nix flavors.
   - **automake**, v1.9.5 or later
     Gnu Automake is a tool for automatically generating make-files.
   - **libtool**, v1.5 or later
     Libtool hides the complexity of developing and using shared libraries for different platforms behind a consistent and portable interface.
   - **doxygen**, any version should do
     Extracts documentation from the C++ source code.
   - **subversion**, any version should do
     Excellent version control tool.

2. Pick up the latest code from the repository with the command:
   svn checkout https://frepple.svn.sourceforge.net/svnroot/frepple/trunk <project_directory>
   More information on working with the Sourceforge svn repositories can be found on http://sourceforge.net/docman/display_doc.php?docid=31070&group_id=1
   The repository allows anonymous connections for checkouts and it is also possible to browse it online from http://frepple.svn.sourceforge.net/viewvc/frepple/
3. Initialize the automake/autoconf/libtool scripts:
   cd <project_directory>
   make -f Makefile.dist prep
   If the command fails you can try the following command to re-initialize all automake/autoconf/libtool scripts to the version you have available on your machine.
   cd <project_directory>
   make -f Makefile.dist prep_force
4. Now the configure script is up to date and you can follow the same steps as in the section Build instructions to compile the code.
5. To refresh your environment with the changes from the repository:
   cd <project_directory>
   svn update
   make -f Makefile.dist prep
   The last command is optional, but still recommended.

2.2.3 VMware virtual machine

A VMware virtual machine is available with a complete demo environment.
It is not intended to be used a production environment.

The setup is based on a Ubuntu Server Linux distribution and has the following main software packages are:

- Linux kernel 2.6.27
- xerces-c 2.8.0
- mysql 5.0.67
- python 2.5.1
- apache httpd 2.2.9
- django 1.0.2
- mod_python 3.3.1
- VMware tools are not installed.

The machines is configured with two CPUs and 500MB of RAM. Update the settings to suit your hardware.

To get up and running:

1. Download and install the VMWare server from http://www.vmware.com/.
2. Download and unzip the virtual machine from the sourceforge site.
3. Using the VMware console open the virtual machine "ubuntu.vmx" and start it.
4. When started the login screen will display the URL where you can browse the demo environment.

5. Instructions about login details, user accounts, database instance, etc will be displayed on the login screen. They are also available in the README.txt file included with the virtual machine.

2.3 Other platforms

FrePPLe hasn't been compiled on any other platforms.

If you succeed in porting the code to another platform, please let us know and give us a hand in updating this document.
In the developer documentation a section is included listing some potential portability issues.

CHAPTER 3

User interface

This chapter describes the Django user interface. The user interface has 3 distinct sub-applications.

1. Data input
   This application is where the input data are maintained.
2. Plan analysis
   This is a collection of reports showing the frePPLe plan output.
3. Execute
   This application is about running tasks in different domains, such as database operations, data loading, creating a new plan, etc...

Each of these applications should be seen as an example reference implementation, rather than a complete and frozen solution. The input data model, the output reports as well as the tasks will need customizing to meet your requirements.

At a later stage, frePPLe will probably package some focussed data models and screens that match certain planning problem, certain industries and/or certain business workflows.

3.1 Data input

The data input application uses the Django admin user interface out-of-the-box.
The default data model maps very closely with the internal representation in the frePPLe engine.

It is pretty easy and straightforward to extend/restructure the data model to match your own domain model.

3.2 Plan analysis

The following reports are currently available:

- **Inventory report**
  The report shows per buffer and per time bucket the inventory profile: the starting inventory, the material consumed, the material produced and the ending inventory.
- **Resource report**
  This report shows the loading of the resources and allows editing the available capacity.
- **Demand report**
  This report shows per item and per time bucket the demand quantity, the supplied quantity and the backlog (as the cumulative gap between the supply and demand).
  A drilldown report is also available to show the detailed data as a list.
- **Forecast report**
  This report provides a convenient way to enter forecast numbers. When entering forecast numbers, the numbers will be disaggregated to the planning buckets.
  A drilldown report is also available to show the detailed data as a list.
- **Operation report**
  The report shows for each time bucket and each operation the quantity started and finished.
  A drilldown report is also available to show the detailed data as a list.
- **Supply Path / Where Used**
  This report follows the bill of material to show how a buffer is being replenished. When called for a resource, it shows the operations using the resource and their supply path.
  The report can't be called directly from the main menu, but it is accessed with a right-click on an entity in the previous reports.
- **Demand Pegging report**
  This report shows how material consumption and material production are matched to each other and associated with independent demands.
  The report can't be called directly from the main menu, but it is accessed with a right-click on a demand.
- **Performance indicator report**
  This report shows some key metrics of the generated plan: number of problems, quantity of demand satisfied, lateness of demands, total inventory, etc...
  The report allows quick review of the plan quality, comparisons between different plans and validation of solver changes.

3.3 Execute

This screen allows you to perform a number of administrative actions and data manipulations.
User permissions will typically be set to limit access to this screen to key users and/or administrators.

The actions in brief that can be performed from this screen:

- Generate a plan.
  This option runs the frePPLe planning engine with the input data from the database. The planning results are exported back into the database.
- Erase the database.
  This will delete all records from the database.
- Load a predefined dataset in the database.
  A number of Django fixtures are available with some demo datasets.
- Generate a model.
  For testing and benchmarking purposes it is extremely useful to be able to generate datasets

with varying sizes and complexity. A few key parameters allow you to create a sample model for such test purposes.

CHAPTER

4

Modeling

This chapter describe the frePPLe data entities, their fields and relationships.

A couple of initial remarks:

- FrePPLe limits itself to the data fields that are relevant for planning.
  An ERP or similar system is more transaction-oriented and will contain plenty of more detailed information.
- The frePPLe data model is designed to be pretty "atomic" in order to be as generic as possible.
  Quite often an entity in a source system will map into a collection or sequence of frePPLe entities.
  For instance, frePPLe doesn't have a model to represent a bill-of-material. Instead the material relations from the BOM are represented as flows on the manufacturing operations.
- The native data format is XML.
  FrePPLe doesn't support namespaces in the XML-data:
  - The XML-data should not be placed in any namespace.
  - FrePPLe XML schema to validate the input data. See the files frepple.xsd and frepple_core.xsd for the definition of the supported constructs.
  - To support subclassing the namespace xsi must be defined as `"http://www.w3.org/2001/XMLSchema-instance"`.

  With the above in mind, the frePPLe XML files typically start with the following lines:

```
<?xml version="1.0" encoding="UTF-8"?>
<plan xmlns:xsi=" http://www.w3.org/2001/XMLSchema-instance">
...
</plan>
```

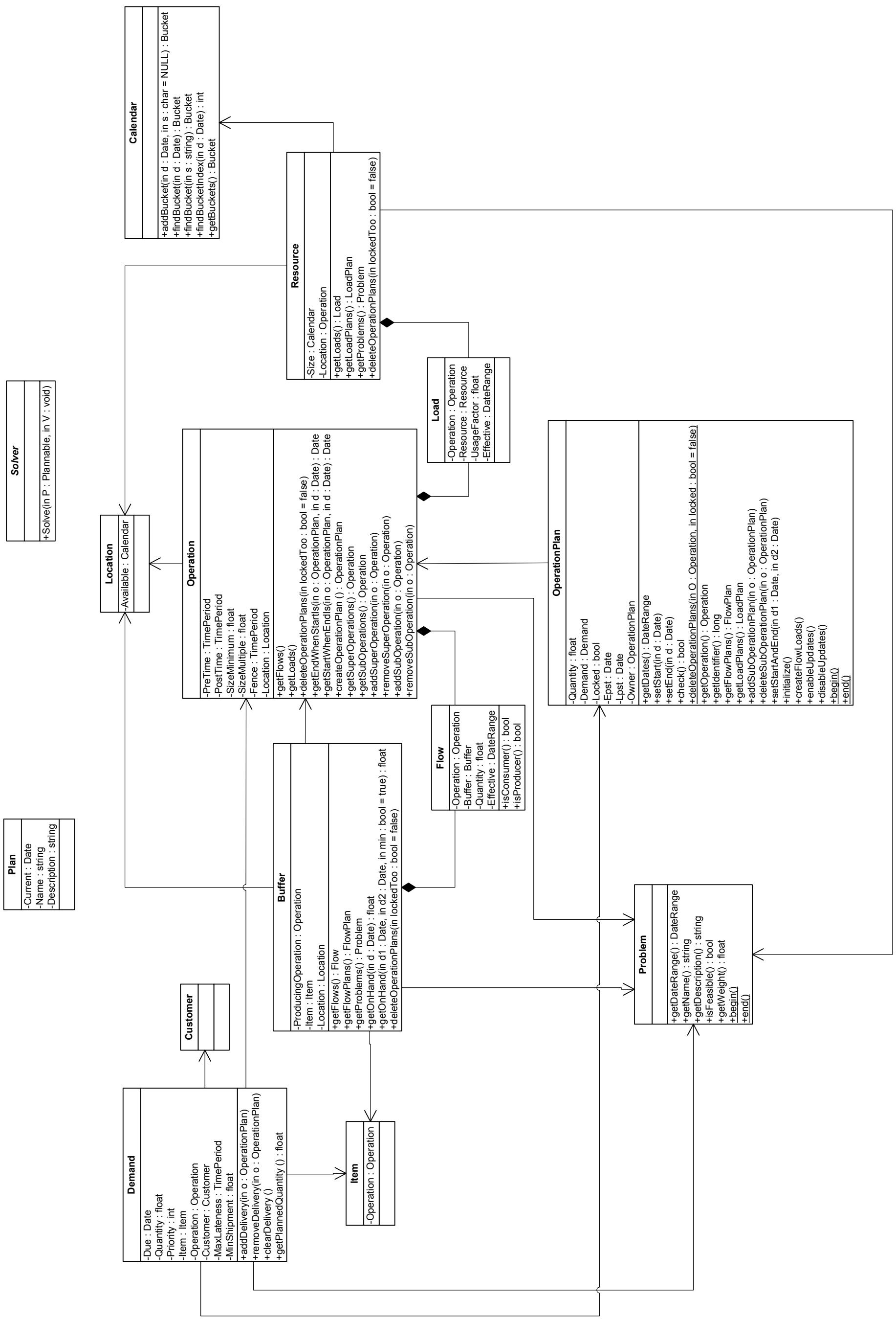
- FrePPLe has a very rich Python API. Detailed programming and scripting of the frePPLe application is possible in this way.
  For complex integration tasks and for customization of the algorithms using Python is the recommended way of working.

- FrePPLe translates input data to the native encoding on your system (as set by the LC_ALL environment variable on Linux, or the code page on Windows).
  The string manipulations inside frePPLe are compatible with the UTF-8 Unicode encoding, but NOT with the UTF-16 or UTF-32 encoding.

Before diving into the details, have a look at the domain model diagram. It shows clear and simple the main entities and their relationships.

1. Domain model
2. Environment variables
3. Python Interpreter
4. Global parameters
5. Buffer
6. Calendar
7. Command
8. Customer
9. Demand
10. Flow
11. Item
12. Load
13. Location
14. Operation
15. OperationPlan
16. Problem
17. Resource
18. Solver

4.1 Domain model


Calendar
+addBucket(in d : Date, in s : char = NULL) : Bucket
+findBucket(in d : Date) : Bucket
+findBucket(in s : string) : Bucket
+findBucketIndex(in d : Date) : int
+getBuckets() : Bucket
Resource
-Size : Calendar
-Location : Operation
+getLoads() : Load
+getLoadPlans() : LoadPlan
+getProblems() : Problem
+deleteOperationPlans(in lockedToo : bool = false)
Solver
+Solve(in P : Plannable, in V : void)
Load
-Operation : Operation
-Resource : Resource
-UsageFactor : float
-Effective : DateRange
Location
-Available : Calendar
Operation
-PreTime : TimePeriod
-PostTime : TimePeriod
-SizeMinimum : float
-SizeMultiple : float
-Fence : TimePeriod
-Location : Location
+getFlows()
+getLoads()
+deleteOperationPlans(in lockedToo : bool = false)
+getEndWhenStartIs(in o : OperationPlan, in d : Date) : Date
+getStartWhenEndIs(in o : OperationPlan, in d : Date) : Date
+createOperPlan () : OperationPlan
+getSuperOperations() : Operation
+getSubOperations() : Operation
+addSuperOperation(in o : Operation)
+removeSuperOperation(in o : Operation)
+addSubOperation(in o : Operation)
+removeSubOperation(in o : Operation)
OperationPlan
-Quantity : float
-Demand : Demand
-Locked : bool
-Epst : Date
-Lpst : Date
-Owner : OperationPlan
+getDates() : DateRange
+setStart(in d : Date)
+setEnd(in d : Date)
+check() : bool
+deleteOperationPlans(in O : Operation, in locked : bool = false)
+getOperation() : Operation
+getIdentifier() : long
+getFlowPlans() : FlowPlan
+getLoadPlans() : LoadPlan
+addSubOperationPlan(in o : OperationPlan)
+deleteSubOperationPlan(in o : OperationPlan)
+setStartAndEnd(in d1 : Date, in d2 : Date)
+initialize()
+createFlowLoads()
+enableUpdates()
+disableUpdates()
+begin()
+end()
Plan
-Current : Date
-Name : string
-Description : string
Flow
-Operation : Operation
-Buffer : Buffer
-Quantity : float
-Effective : DateRange
+isConsumer() : bool
+isProducer() : bool
Buffer
-ProducingOperation : Operation
-Item : Item
-Location : Location
+getFlows() : Flow
+getFlowPlans() : FlowPlan
+getProblems() : Problem
+getOnHand(in d : Date) : float
+getOnHand(in d1 : Date, in d2 : Date, in min : bool = true) : float
+deleteOperationPlans(in lockedToo : bool = false)
Problem
+getDateRange() : DateRange
+getName() : string
+getDescible() : string
+isFeasible() : bool
+getWeight() : float
+begin()
+end()
Customer
Demand
-Due : Date
-Quantity : float
-Priority : int
-Item : Item
-Operation : Operation
-Customer : Customer
-MaxLateness : TimePeriod
-MinShipment : float
+addDelivery(in o : OperationPlan)
+removeDelivery(in o : OperationPlan)
+clearDelivery ()
+getPlannedQuantity () : float
Item
-Operation : Operation

4.2 Environment variables

A number of environment variables influence frePPLe.

Variable	Description
FREPPLE_HOME	FrePPLe uses the following configuration files: • The file *frepple.xsd* points to the xsd schema for the frePPLe XML files. This xsd file typically references additional xsd files located in the same directory. • If present, the commands in the file *init.xml* are executed automatically when frePPLe is started. This is the recommended place to load the modules you need. • Plugin *module libraries*. FrePPLe searches the following directories in sequence to locate these files. • The current directory. • The directory pointed to by the FREPPLE_HOME environment variable. • The data directory where the default configuration files are installed. This applies only for Linux and Unix platforms. • The library directory where the default module libraries are installed. This applies only for Linux and Unix platforms. • For the loading module libraries frePPLe also searches the standard path for location shared libraries. Configuring this is platform dependent. By setting the FREPPLE_HOME environment variable you can easily configure the application to to use your override files.
LC_ALL	FrePPLe stores string data internally using the encoding associated with your locale. This setting is important when dealing with non-ascii characters in your data. Your locale needs to support all characters being used, just as your database will also need to support them. **Most modern Linux distributions have a default locale that supports utf-8, which allows every possible unicode character to be represented. On Windows, this environment variable isn't used and frePPLe can only represent characters present in the default windows code page.**

TZ	FrePPLe uses the C-library functions for date and time manipulations. These functions are respecting timezones and daylight saving time. **Especially the daylight saving time of your timezone can give some unexpected results: twice a year you'll find a day with 25 or 23 hours. To disable any effects of daylight saving time, change the TZ variable to a timezone without daylight saving time, e.g. 'EST'.**
NUMBER_OF_PROCESSORS	Controls the maximum number of parallel threads that can be used for a frePPLe command. On windows platforms, this variable is automatically set to the number of cpu's and cores of your machine. On other platforms it'll need to be set explicitly. When left unspecified, a default value of 1 is used: i.e. sequential, single-threaded execution.

4.3 Python Interpreter

FrePPle comes with an embedded interpreter for the Python language.
The full capabilities of this scripting language are accessible from frePPLe, and Python also has access to the frePPLe objects in memory.
Python is thus a very powerful way to interact with frePPLe.

The Python 2.5.x language needs to be installed on your computer. When you compile the module from source, version 2.4.x will also work.
The versions 2.6.x and 3.x have not been tested with frePPLe.

Python code can be implemented as a command (w#COMMAND_PYTHON) or as a processing instruction (p 27).

When the frePPLe library is initialized it searches for the presence of a file *init.py*. If it is found its Python code is executed.
This provides a clean mechanism to define global Python functions and classes.

4.3.1 Processing instruction python

Python code can also be included as a XML processing instruction.

Note that the processing instruction is executed when the XML file is parsed. Commands on the other hand are executed AFTER the parsing the complete XML data.

Example XML structure:

```
<plan>
<?python
   def MyFunction():
```

```
      print "Hello World"
 ?>
    <commands>
      <command xsi:type="command_python" cmdline="MyFunction()" />
    </commands>
 </plan>
```

4.4 Global parameters

A number of global settings and parameters are described here.

4.4.1 Fields

Field	Type	Description
name	normalizedString	Model name. Default is null.
description	string	Free format description.
current	dateTime	The ‘now’ date for the plan. It distinguishes the past from the future.
logfile	normalizedString	File name where all output will be sent to. If left unspecified, the output appears on the standard output. If the filename starts with ‘+’ an existing logfile with the same name is being appended to, instead of being overwritten.

4.4.2 Example XML structures:

- Global initialization section

```
<plan>
   <name>Demo model</name>
   <description>A demo model demonstrating frePPLe</description>
   <current>2007-01-01T00:00:00</current>
   <logfile>frepple.log</logfile>
</plan>
```

4.4.3 Example Python code:

- Global initialization section

```
frepple.settings.name = "Plan name"
frepple.settings.description = "Plan description"
frepple.settings.current = datetime.datetime(2007,1,1)
frepple.settings.logfile = "frepple.log"
```

4.5 Buffer

A buffer is a storage for a item.
It represents a place where inventory of an item is kept.

Different types of buffers exist:

- buffer_default (p 31):
  The default buffer uses an "producing" operation to replenish it with additional material.
- buffer_procure (p 31):
  A buffer that is replenished by a supplier. A number of parameters control the re-ordering policy: classic re-order point, fixed time ordering, fixed quantity ordering, etc...
- buffer_infinite (p 32):
  An infinite buffer has an infinite supply of the material is available.

4.5.1 Fields

Field	Type	Description
name	non-empty string	Name of the buffer. This is the key field and a required attribute.
description	string	Free format description.
category	normalizedString	Free format category.
subcategory	normalizedString	Free format subcategory.
owner	buffer	Buffers can be organized in a hierarchical tree. This field defines the parent buffer. No specific planning behavior are currently linked to such a hierarchy.
members	list of buffer	Buffers can be organized in a hierarchical tree. This field defines a list of child buffers.
location	location	Location of the buffer. Default is null. The working hours and holidays for the buffer are taken from the 'available' calendar of the location.
item	item	Item being stored in the buffer. Default is null.
onhand	double	Inventory level at the start of the time horizon. Default is 0.
carrying_cost	double	The cost of carrying inventory in this buffer. The value is an annual percentage of the item sales price. The default value is 1.0.

minimum	calendar	Refers to a calendar storing the desired minimum inventory level, aka safety stock. The solver treats this as a soft constraint, ie it tries to meet this inventory level but will go below the minimum level if required to meet the demand. A problem is reported when the inventory drops below this level. The safety stock target is expressed as a quantity. If you want to define a safety stock target as a time value, you can set a post-operation time on the producing operation of a buffer.
maximum	calendar	Refers to a calendar storing the maximum inventory level. This field is not used by the solver. A problem is reported when the inventory level is higher than this limit.
producing	operation	This operation will be instantiated by the solver to replenish the buffer with additional material.
detectproblems	boolean	Set this field to false to supress problem detection on this buffer. Default is true.
flows	list of flow	Defines material flows consuming from or producing into this buffer.
flowplans	list of flowplan	This field is populated during an export with the plan results for this buffer. It shows the complete inventory profile. The field is export-only.
level	integer	Indication of how upstream/downstream this entity is situated in the supply chain. Lower numbers indicate the entity is close to the end item, while a high number will be shown for components nested deep in a bill of material. The field is export-only.
cluster	integer	The network of entities can be partitioned in completely independent parts. This field gives the index for the partition this entity belongs to. The field is export-only.
action	A C AC (default) R	Type of action to be executed: • A: Add an new entity, and report an error if the entity already exists. • C: Change an existing entity, and report an error if the entity doesn't exist yet. • AC: Change an entity or create a new one if it doesn't exist yet. • R: Remove an entity, and report an error if the entity doesn't exist.

4.5.2 buffer_default

The default buffer uses an "producing" operation to replenish it.

No fields are defined in addition to the ones listed above.

4.5.3 buffer_procure

A procurement buffer is replenished by a supplier.

A number of parameters control the re-ordering policy: classic re-order point, fixed time ordering, fixed quantity ordering, etc...
The parameters LEADTIME, MININVENTORY and MAXINVENTORY define a replenishment with a classical re-orderpoint policy. The inventory profile will show the typical sawtooth shape.
The parameters MININTERVAL and MAXINTERVAL put limits on the frequency of replenishments. The inventory profile will have "teeth" of variable size but with a controlled interval.
The parameters SIZE_MINIMUM, SIZE_MAXIMUM and SIZE_MULTIPLE put limits on the size of the replenishments. The inventory profile will have "teeth" of controlled size but with variable intervals.
Playing with these parameters allows flexible and smart procurement policies to be modelled.

Note that frePPLe doesn't include any logic to set these parameters in an optimal way. The parameters are to be generated externally and frePPLe only executes based on the parameter settings.
At a later stage a module to compute these parameters could be added.

The PRODUCING field is unused for this buffer type.
Propagation through a bill of material will be stopped at a procurement buffer.

Field	**Type**	**Description**
leadtime	duration	Time taken between placing the purchase order with the supplier and the delivery of the material. When the "LEADTIME" constraint is enabled in the solver, it won't create any new procurement orders that would need to start in the past.
fence	duration	Time window (from the current date of the plan) during which procurement orders are expected to be released. When the "FENCE" constraint is enabled in the solver, it won't create any new operation plans in this time fence. Only the externally supplied existing procurement plans will then exist in this time window.
mininventory	Positive double	Inventory level triggering a new replenishment. The actual inventory can drop below this value.
maxinventory	Positive double	Inventory level to which we try to replenish. The actual inventory can exceed this value.

mininterval	duration	Minimum time between replenishments. The order quantity will be increased such that it covers at least the demand in the minimum interval period. The actual inventory can exceed the target set by the MinimumInventory parameter.
maxinterval	duration	Maximum time between replenishments. The order quantity will replenish to an inventory value less than the maximum when this maximum interval is reached.
size_minimum	Positive double	Minimum quantity for a replenishment. This parameter can cause the actual inventory to exceed the target set by the MinimumInventory parameter.
size_maximum	Positive double	Maximum quantity for a replenishment. This parameter can cause the maximum inventory target never to be reached.
size_multiple	Positive double	All replenishments are rounded up to a multiple of this value.

4.5.4 buffer_infinite

An infinite buffer has an infinite supply of the material is available.

The PRODUCING field is unused for this buffer type.
Propagation through a bill of material will be stopped at an infinite buffer.

4.5.5 Example XML structures:

- Adding or changing a buffer

```
<plan>
  <buffers>
    <buffer name="item a @ location b">
      <item name="item a" />
      <location name="location b" />
      <onhand>10</onhand>
    </buffer>
  </buffers>
</plan>
```

- Update the current inventory information of an existing buffer

```
<plan>
  <buffers>
    <buffer name="item a @ location b" onhand="100"  action="C" />
  </buffers>
</plan>
```

- Deleting a buffer

```
<plan>
   <buffers>
      <buffer name="item a @ location b" action="R"/>
   </buffers>
</plan>
```

4.5.6 Example Python code:

- Adding or changing a buffer

```
it = frepple.item(name="item a")
loc = frepple.location(name="location b")
buf = frepple.buffer(name="item a @ location b",
        onhand=10, item=it, location=loc)
```

- Update the current inventory information of an existing buffer

```
buf = frepple.buffer(name="item a @ location b",
        onhand=10, action="C")
```

- Deleting a buffer

```
buf = frepple.buffer(name="item a @ location b", action="R")
```

- Iterate over buffers, flows and flowplans

```
for b in frepple.buffers():
  print "Buffer:", b.name, b.description, b.category,
  for l in b.flows:
    print " Flow:", l.operation.name, l.quantity,
      l.effective_start, l.effective_end
  for l in b.flowplans:
    print " Flowplan:", l.operationplan.operation.name,
      l.quantity, l.date
```

4.6 Calendar

A calendar represents a value that is varying over time.
Calendars can be linked to multiple entities: a maximum capacity limit of a resource, a minimum capacity usage of a resource, a minimum or maximum inventory limit of a buffer, etc…

Different types of calendar exist:

- calendar_void:
  A calendar without any value in its buckets.
- calendar_double:
  A calendar storing double numbers.
- calendar_integer:
  A calendar storing integer numbers.
- calendar_boolean:
  A calendar storing boolean values.

- calendar_string:
  A calendar storing string values.
- calendar_operation:
  A calendar storing operation values.

A calendar has multiple buckets to define the values over time. To determine the calendar value at a certain date the calendar will evaluate each of the buckets and combine the results in the following way:

- A bucket is only valid from its "start" date (inclusive) till its "end" date (exclusive).
  Outside of this date range a bucket is never selected.
- If multiple bucket are effective on a date, the one with the lowest "priority" value is taken.
  In case buckets have the same priority, the value of the bucket with the latest start date is selected.
- In case no bucket is effective on a certain date, the calendar will return the "default" value.

4.6.1 Calendar Fields

Field	Type	Description
name	non-empty string	Name of the calendar. This is the key field and a required attribute.
default	Varies with the calendar type	The default value of the calendar when no bucket is effective.
buckets	List of bucket	A list of a buckets.
action	A C AC (default) R	Type of action to be executed: • A: Add an new entity, and report an error if the entity already exists. • C: Change an existing entity, and report an error if the entity doesn't exist yet. • AC: Change an entity or create a new one if it doesn't exist yet. • R: Remove an entity, and report an error if the entity doesn't exist.

4.6.2 Bucket Fields

Field	Type	Description
start	dateTime	Start date of the validity of this bucket. When left unspecified, the entry is effective from the infinite past.
end	dateTime	End date of the validity of this bucket. When left unspecified, the entry is effective indefinately in the future.

name	normalizedString	Optional name of the bucket. When left unspecified the default name is the start date of the bucket.
priority	integer	Priority of this bucket when multiple buckets are effective for the same date. Lower values indicate higher priority.
value	Varies with the calendar type	The actual time-varying value.
action	A C AC (default) R	Type of action to be executed: • A: Add an new entity, and report an error if the entity already exists. • C: Change an existing entity, and report an error if the entity doesn't exist yet. • AC: Change an entity or create a new one if it doesn't exist yet. • R: Remove an entity, and report an error if the entity doesn't exist.

4.6.3 Example XML structures:

- Adding or changing a calendar and its buckets

```
<plan>
  <calendars>
    <calendar name="cal" xsi:type="calendar_double">
      <default>5</default>
      <buckets>
        <bucket start="2007-01-01T00:00:00" value="10"
           priority="1"/>
        <!-- This entry overrides the first one during February. →
        <bucket start="2007-02-01T00:00:00" end="2007-03-01T00:00:00
           value="20" priority="0"/>
      </buckets>
    </calendar >
  </calendars>
</plan>
```

- Removing a calendar

```
<plan>
   <calendars>
      <calendar name="cal" action="R"/>
   </calendars>
</plan>
```

4.6.4 Example Python code:

- Adding or changing a calendar and its buckets

```
cal = frepple.calendar_double(name="cal", default=5)
```

- Removing a calendar

```
frepple.calendar(name="cal", action="R")
```

4.7 Command

All state changes in frePPLe are modeled as commands.

Commands are read from XML input, and executed **at the end** of parsing/processing all input. Commands are read and executed, but are never exported or saved again.

The XML interface for commands is deprecated and will be removed in a next release. All commands will be defined and executed through the Python interface only.

A wide range of commands exists to control the application:

- command_python (p 36) allows to execute Python code in the embedded interpreter.
- command_list (p 37) groups a number of commands, which can be executed in sequence or in parallel.
- command_loadlib (p 38) dynamically loads an extension module.
- command_system (p 38) executes a operating system command.
- command_readxml (p 39) processes a XML-file from the local file system.
- command_readxmlstring (p 39) processes a XML-formatted string.
- command_setenv (p 40) updates an environment variable.
- command_erase (p 41) removes part of the model or plan from memory.
- command_save (p 41) saves the model to an XML-formatted file.
- command_saveplan (p 42) saves the most important plan information to a file.
- command_size (p 43) prints information about the memory size of the model and other system parameters.
- command_solve (p 43) runs a solver.

4.7.1 command_python

The command allows you to run Python code in the embedded interpreter. You can specify the Python code directly, or provide the name of a file containing the code.
The interpreter can execute generic scripts, and it also has access to the frePPLe objects.

The interpreter is multi-threaded. Multiple python scripts can run in parallel. However, Python internally executes only one thread at a time and the interpreter switches between the active threads.

A single, global interpreter instance is used. A global Python variable or function is thus visible across multiple invocations of the Python interpreter.

Field	Type	Description
cmdline	String	Python command to be executed.
filename	normalizedString	Filename with Python commands to be executed. When both the CMDLINE and FILENAME fields are filled in only the CMDLINE Python code will be executed.
verbose	boolean	Echo information about the command execution in the log.

Example XML structure:

```
<plan>
   <commands>
     <command xsi:type="command_python"
        cmdline="print 'Hello World'" />
   </commands>
</plan>
```

4.7.2 command_list

This command groups a number of commands, which can be executed in sequence or in parallel.

Field	Type	Description
command	command	The sub-commands part of this list.Multiple sub-commands can be defined.
abortonerror	boolean	When executing commands sequentially, this field specifies the behavior in the case of an error: • When set to false, the execution will simply continue with the next command. • When set to true, the execution of the list will be aborted. The default is true.
maxparallel	Positive integer	Maximum number of commands to be executed in parallel. The default value is 1, ie sequential execution.
verbose	boolean	Echo information about the command execution in the log. This field is inherited by the sub-commands.

Example XML structure:

```
<plan>
   <commands>
     <verbose>true</verbose>
     <command xsi-type="command_list" maxparallel="100">
```

```
        <command xsi:type="command_system"
           cmdline="sleep 1 && echo  after 1 second" />
        <command xsi:type="command_system"
           cmdline="sleep 2 && echo  after 2 second" />
      </command>
    </commands>
</plan>
```

4.7.3 command_loadlib

This command dynamically loads an extension module.

Field	**Type**	**Description**
filename	normalizedString	Name of the shared library file to be loaded. The operating system should allow frePPLe to locate the file. The directories listed in the following environment variable should include the module shared library. • LD_LIBRARY_PATH variable for Linux, Solaris • LIBPATH for AIX • SHLIB_PATH for HPUX • PATH for windows and cygwin
parameter	parameter	Initialization and configuration values that are passed to the module's initialization routine. A parameter consists of a PARAMETER and VALUE pair, as shown in the example below.
verbose	boolean	Echo information about the command execution in the log.

Example XML structure:

```
<plan>
    <commands>
      <verbose>true</verbose>
      <command xsi:type="command_loadlib" filename="mod_python.so" />
      <commandxsi:type="command_loadlib" filename="your_module.so">
         <parameter name="test1" value="val1"/>
         <parameter>
           <name>test2</name>
           <value>val2</value>
         </parameter>
      </command>
    </commands>
</plan>
```

4.7.4 command_system

Executes a operating system command in seperate process.

Field	Type	Description
cmdline	string	Command line to be executed in an operating shell.
verbose	boolean	Echo information about the command execution in the log.

Example XML structure:

```
<plan>
   <commands>
     <verbose>true</verbose>
     <command xsi:type="command_system" cmdline="sleep 1" />
     <command xsi:type="command_system" cmdline="do_something.sh" />
   </commands>
</plan>
```

4.7.5 command_readxml

This command reads and processes a XML-file from the local file system.

Field	Type	Description
filename	normalizedString	Name of the data file to be loaded.
validate	boolean	When set to true, the XML data are validated against the XML-schema. The default value is true, for security reasons. When parsing large files with a trusted structure setting this field to false will speed up the import.
verbose	boolean	Echo information about the command execution in the log.

Example XML structure:

```
<plan>
   <commands>
     <command xsi:type="command_readxml" filename="input.xml" />
   </commands>
</plan>
```

4.7.6 command_readxmlstring

This command processes a XML-formatted data string.

Field	Type	Description
data	string	XML-formatted data to be processed.
validate	boolean	When set to true, the XML data are validated against the XML-schema. The default value is true, for security reasons. When processing large data strings with a trusted structure setting this field to false will speed up the execution.
verbose	boolean	Echo information about the command execution in the log.

Example XML structure:

```
<plan>
  <commands>
    <commandxsi:type="command_readxmlstring">
      <data>
        <![CDATA[
 <plan xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <locations>
 <location name="Location 1" action="R"/>
 </locations>
 </plan>
 ]]>
      </data>
    </command>
  </commands>
</plan>
```

4.7.7 command_setenv

This command updates an environment variable.

Field	Type	Description
variable	normalizedString	Environment variable to be updated.
value	string	New value of the variable.
verbose	boolean	Echo information about the command execution in the log.

Example XML structure:

```
<plan>
  <commands>
    <command xsi:type="command_setenv" variable="VAR1" value="VAL1" />
    <!-- Showing the variables in a shell command. →
    <command xsi:type="command_system" cmdline="echo ${VAR1}" />
  </commands>
</plan>
```

4.7.8 command_erase

Use this command to erase the plan or the entire model from memory.

Field	Type	Description
mode	Plan Model	When set to "model" the complete model is erased. You will again have a completely empty model. When set to "plan" only the plan information is erased, ie all operationplans with their load- and flowplans are removed (except the ones that are locked).
verbose	boolean	Echo information about the command execution in the log.

Example XML structure:

```
<plan>
  <commands>
    <command xsi:type="command_erase" mode="plan" />
  </commands>
</plan>
```

4.7.9 command_save

This commands saves the model into an XML-formatted file.

Field	Type	Description
filename	normalizedString	Name of the output file.

content	STANDARD PLAN PLANDETAIL	Controls the level of detail in the output: • STANDARD plan information is sufficient for restoring the model from the output file.\\ This is the default mode. • PLAN adds more detail about its plan with each entity. A buffer will report on its flowplans, a resource reports on its loadplans, and a demand on its delivery operationplans. • PLANDETAIL goes even further and includes full pegging information the output. A buffer will report how the material is supplied and which demands it satisfies, a resource will report on how the capacity used links to the demands, and a demand shows the complete supply path used to meet it.
headerstart	string	The first line of the XML output. The default value is: `<?xml version="1.0" encoding="UTF-8"?>`
headeratts	string	Predefined attributes of the XML root-element. The default value is: `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
verbose	boolean	Echo information about the command execution in the log.

Example XML structure:

```
<plan>
  <commands>
    <command xsi:type="command_save" filename="output.xml" />
  </commands>
</plan>
```

4.7.10 command_saveplan

This command saves the most important plan information to a file.
It is used for the unit tests, but its' usefullness in a real-life implementation is probably limited.

Field	Type	Description
filename	normalizedString	Name of the output file.
verbose	boolean	Echo information about the command execution in the log.

Example XML structure:

```
<plan>
  <commands>
```

```
    <command xsi:type="command_saveplan" />
      <filename>output.xml</filename>
    </command>
  </commands>
</plan>
```

4.7.11 command_size

This command prints information about the memory size of the model and other sytem parameters.

Field	Type	Description
verbose	boolean	Echo information about the command execution in the log.

Example XML structure:

```
<plan>
   <commands>
     <command xsi:type="command_size" />
   </commands>
</plan>
```

4.7.12 command_solve

This command will execute a solver.

Field	Type	Description
solver	solver	Points to the solver to execute.
verbose	boolean	Echo information about the command execution in the log. Note that the solver itself uses its LOGLEVEL field to control the amount of information to write about its' progress.

Example XML structure:

```
<plan>
   <commands>
     <verbose>true</verbose>
     <command xsi:type="command_solve">
       <solver xsi:type="solver_mrp" name="MRP"
         constraints="7" loglevel="2" />
     </COMMAND>
   </commands>
</plan>
```

4.8 Customer

Demands are associated with a customer.

Customers can be organized in a hierarchical tree to represent the sales organization's structure.

FrePPLe uses customers only from reporting purposes, no real planning logic is currently linked to them.

4.8.1 Fields

Field	Type	Description
name	non-empty string	Name of the customer. This is the key field and a required attribute.
description	string	Free format description.
category	normalizedString	Free format category.
subcategory	normalizedString	Free format subcategory.
owner	customer	Customers are organized in a hierarchical tree. This field defines the parent customer.
members	list of customer	Customers are organized in a hierarchical tree. This field defines a list of child customer.
action	A C AC (default) R	Type of action to be executed: • A: Add an new entity, and report an error if the entity already exists. • C: Change an existing entity, and report an error if the entity doesn't exist yet. • AC: Change an entity or create a new one if it doesn't exist yet. • R: Remove an entity, and report an error if the entity doesn't exist.

4.8.2 Example XML structures:

- Adding or changing a customer

```
<plan>
   <customers>
      <customer name="customer A" category="Direct"/>
   </customers>
</plan>
```

- Deleting a customer

```
<plan>
   <customers>
      <customer name="customer A" action="R"/>
   </customers>
</plan>
```

4.8.3 Example Python code:

- Adding or changing a customer

```
cust = frepple.customer(name="customer A", category="Direct")
```

- Deleting a customer

```
cust = frepple.customer(name="customer A", action="R")
```

4.9 Demand

Define independent demands for items.
These can be actual customer orders, or forecasted demands.

4.9.1 Fields

Field	Type	Description
name	non-empty string	Name of the demand. This is the key field and a required attribute.
description	string	Free format description.
category	normalizedString	Free format category.
subcategory	normalizedString	Free format subcategory.
owner	demand	Demands are organized in a hierarchical tree. This field defines the parent demand.
members	list of demand	Demands are organized in a hierarchical tree. This field defines a list of child demand.
quantity	double	Requested quantity.
item	item	Requested item.
due	dateTime	Due date of the demand.
priority	integer	Priority of the demand relative to the other demands. A lower number indicates higher priority. The default value is 0.

operation	operation	Operation to be used to satisfy the demand. If left unspecified the operation on the item will be used.
customer	customer	Customer placing the demand.
detectproblems	boolean	Set this field to false to supress problem detection on this demand. Default is true.
maxlateness	duration	The maximum delay that can be accepted to satisfy this demand. The default value allows an infinite delay.
minshipment	Positive double	The minimum quantity allowed for the shipment operationplans that satisfy this demand. The default is 0, allowing deliveries of any size.
action	A C AC (default) R	Type of action to be executed: • A: Add an new entity, and report an error if the entity already exists. • C: Change an existing entity, and report an error if the entity doesn't exist yet. • AC: Change an entity or create a new one if it doesn't exist yet. • R: Remove an entity, and report an error if the entity doesn't exist.

4.9.2 Example XML structures:

- Adding or changing demands

```
<plan>
  <demands>
   <demand name="order A">
     <quantity>10</quantity>
     <due>2007-01-10T00:00:00</due>
     <priority>1</priority>
     <item name="item 1" />
     <!-- Don’t allow any delay →
     <maxlateness>P0D</maxlateness>
     <!-- Don’t create a delivery for less than 5 units →
     <minshipment>5</minshipment>
   </demand>
   <demand name="order B" quantity="10"
       due="2007-01-10T00:00:00" priority="1" >
     <item name="item 1" />
   </demand>
  </demands>
</plan>
```

- Removing a demand

```
<plan>
  <demands>
      <demand name="order ABC" action="R"/>
  </demands>
</plan>
```

4.9.3 Example Python code:

- Adding or changing demands

```
it = frepple.item(name="item 1")
dem1 = frepple.demand(name="order A", quantity=10,
  due=datetime.datetime(2007,01,10), priority=1, item=it,
  # Don't allow any delay
  maxlateness=0,
  # Don't create a delivery for less than 5 units
  minshipment=5)
dem2 = frepple.demand(name="order B", quantity=10,
    due=datetime.datetime(2007,1,10), priority=1", item=it)
```

- Removing a demand

```
frepple.demand(name="order ABC", action="R")
```

- Iterating over all demands and their deliveries

```
for d in frepple.demands():
  print "Demand:", d.name, d.due, d.item.name, d.quantity
  for i in d.operationplans:
    print "  Operationplan:", i.operation.name, i.quantity, i.end
```

4.10 Flow

Flows are used to model the consumption and production of material from buffers.

Two types of flows exist:

- FLOW_START: Flows that consume material at the start of an operationplan.
- FLOW_END: Flows that produce material at the end of an operationplan.

4.10.1 Fields

Field	Type	Description
buffer	buffer	Buffer from which material will be moved or transferred into. This is a required field.
operation	operation	Operation to which the material flow is associated. This is a required field.

quantity	double	Material quantity being consumed or produced per unit of the operationplan.
effective_start	dateTime	Date after which the material consumption is valid. Before this date the planned quantity is always 0.
effective_end	dateTime	Date at which the material consumption becomes invalid. After this date (and also at the exact date) the planned quantity is always 0.
action	A C AC (default) R	Type of action to be executed: • A: Add an new entity, and report an error if the entity already exists. • C: Change an existing entity, and report an error if the entity doesn't exist yet. • AC: Change an entity or create a new one if it doesn't exist yet. • R: Remove an entity, and report an error if the entity doesn't exist.

4.10.2 Example XML structures:

- Defining a flow

```
<plan>
   <flows>
      <flow xsi:type="flow_start">
        <buffer name="buffer component"/>
        <operation name="operation B"/>
        <quantity>-2</quantity>
      </flow>
   </flows>
</plan>
```

- Defining a flow nested in an operation structure

```
<plan>
   <operations>
     <operation name="operation B">
       <flows>
         <flow xsi:type="flow_start">
           <buffer name="buffer component"/>
           <quantity>-2</quantity>
         </flow>
         <flow xsi:type="flow_end">
           <buffer name="buffer end item"/>
           <quantity>1</quantity>
         </flow>
       </flows>
     </operation>
```

```
  </operations>
</plan>
```

- Defining a flow nested in a buffer structure

```
<plan>
  <buffers>
    <buffer name="buffer component">
      <flows>
        <flow xsi:type="flow_start">
          <operation name="operation A"/>
          <quantity>-2</quantity>
        </flow>
        <flow xsi:type="flow_start">
          <operation name="operation B"/>
          <quantity>-1</quantity>
        </flow>
      </flows>
    </buffer>
  </buffers>
</plan>
```

- Deleting a flow

```
<plan>
  <flows>
    <flow action="R">
      <buffer name="buffer component"/>
      <operation name="operation B"/>
    </flow>
  </flows>
</plan>
```

4.11 Item

An item represents an end product, intermediate product or a raw material.

Each demand is associated with an item.

A buffer is also associated with an item: it represents a storage of the item.

4.11.1 Fields

Field	Type	Description
name	non-empty string	Name of the item. This is the key field and a required attribute.
description	string	Free format description.

category	normalizedString	Free format category.
subcategory	normalizedString	Free format subcategory.
owner	item	Items are organized in a hierarchical tree. This field defines the parent item.
members	list of item	Items are organized in a hierarchical tree. This field defines a list of child items.
operation	operation	This is the operation used to satisfy a demand for this item. If left unspecified the value is inherited from the parent item. See also the OPERATION field on the DEMAND.
price	double	Cost or price of the item. Depending on the precise usage and business goal it should be evaluated which cost to load into this field: purchase cost, booking value, selling price… For most applications the booking value is the appropriate one.
action	A C AC (default) R	Type of action to be executed: • A: Add an new entity, and report an error if the entity already exists. • C: Change an existing entity, and report an error if the entity doesn't exist yet. • AC: Change an entity or create a new one if it doesn't exist yet. • R: Remove an entity, and report an error if the entity doesn't exist.

4.11.2 Example XML structures:

- Adding or changing an item and its delivery operation

```
<plan>
  <items>
    <item name="item A">
      <operation name="Delivery of item A"
       xsi:type="operation_fixed_time">
        <duration>24:00:00</duration>
      </operation>
      <owner name="Item class A"/>
    </item>
  </items>
</plan>
```

- Deleting an item

```
<plan>
   <items>
      <item name="item A" action="R"/>
```

```
  </items>
</plan>
```

4.11.3 Example Python code:

- Adding or changing an item and its delivery operation

```
oper = frepple.operation_fixed_time(name="Deliver item A",
         duration=24*3600)
it1 = frepple.item(name="Item class A")
it2 = frepple.item(name="item A", operation=oper, owner=it1)
```

- Deleting an item

```
frepple.item(name="item A", action="R")
```

4.12 Load

Loads are used to model the capacity consumption of an operation.

4.12.1 Fields

Field	Type	Description
resource	resource	Resource being loaded. This is a required field.
operation	operation	Operation loading the resource. This is a required field.
quantity	double	Load factor of the resource. The default value is 1.0.
effective_start	dateTime	Date after which the resource load is valid. Before this date the planned quantity is always 0.
effective_end	dateTime	Date at which the resource load becomes invalid. After this date (and also at the exact date) the planned quantity is always 0.
action	A C AC (default) R	Type of action to be executed: • A: Add an new entity, and report an error if the entity already exists. • C: Change an existing entity, and report an error if the entity doesn't exist yet. • AC: Change an entity or create a new one if it doesn't exist yet. • R: Remove an entity, and report an error if the entity doesn't exist.

4.12.2 Example XML structures:

- Defining a load

```
<plan>
   <loads>
      <load>
        <resource name="machine A"/>
        <operation name="operation B"/>
      </load>
   </loads>
</plan>
```

- Defining a load nested in an operation structure

```
<plan>
   <operations>
     <operation name="operation B">
       <loads>
         <load>
           <resource name="machine A"/>
           <usage>1</usage>
         </load>
       </loads>
     </operation>
   </operations>
</plan>
```

- Defining a load nested in a resource structure

```
<plan>
   <resources>
     <resource name="machine A">
       <loads>
         <load>
           <operation name="operation B"/>
           <usage>2</usage>
         </load>
         <load>
           <operation name="operation C"/>
           <usage>1</usage>
         </load>
       </loads>
     </resource>
   </resources>
</plan>
```

- Deleting a load

```
<plan>
   <loads>
      <load action="R">
        <resource name="machine A"/>
        <operation name="operation B"/>
      </load>
   </loads>
</plan>
```

4.13 Location

A location is a (physical or logical) place where resources, buffers and operations are located.

FrePPLe uses locations from reporting purposes, and the ‘available’ calendar is used to model the working hours and holidays of resources, buffers and operations.

4.13.1 Fields

Field	**Type**	**Description**
name	non-empty string	Name of the location. This is the key field and a required attribute.
description	string	Free format description.
category	normalizedString	Free format category.
subcategory	normalizedString	Free format subcategory.
available	calendar_boolean	A calendar that defines the working hours and holidays for the location. All operations, buffers and resources at this location will use it.
owner	location	Locations are organized in a hierarchical tree. This field defines the parent location.
members	list of location	Locations are organized in a hierarchical tree. This field defines a list of child locations.
action	A C AC (default) R	Type of action to be executed: • A: Add an new entity, and report an error if the entity already exists. • C: Change an existing entity, and report an error if the entity doesn’t exist yet. • AC: Change an entity or create a new one if it doesn’t exist yet. • R: Remove an entity, and report an error if the entity doesn’t exist.

4.13.2 Example XML structures:

- Adding or changing a location

```
<plan>
   <locations>
      <location name="site A">
        <category>cat A</category>
        <owner name="Manufacturing sites"/>
      </location>
   </locations>
</plan>
```

- Alternate format of the previous example

```
<plan>
   <locations>
      <location name="Manufacturing sites">
        <members>
          <location name="site A" category="cat A"/>
        </members>
      </location>
   </locations>
</plan>
```

- Deleting a location

```
<plan>
   <locations>
      <location name="site A" action="R"/>
   </locations>
</plan>
```

4.13.3 Example Python code:

- Adding or changing a location

```
loc1 = frepple.location(name="Manufacturing sites")
loc2 = frepple.location(name="site A", category="cat A", owner=loc1)
```

- Deleting a location

```
frepple.location(name="site A", action="R")
```

4.14 Operation

An operation represents an activity: these consume and produce material, take time and also require capacity.

An operation consumes and produces material, modeled through flows.

An operation requires capacity, modeled through loads.

Different operation types exist:

- operation_fixed_time (p 57):
  Models an operation with a duration that is independent of the quantity. A good example is a transport or a procurement operation.
- operation_time_per (p 57):
  Models an operation where the duration increases linear with the quantity. A good example is a manufacturing operation where the duration is determined by the production rate of a machine.
- operation_alternate (p 57):
  Models a choice between different operations.
- operation_routing (p 58):
  Models a sequence a number of ‘step’ sub-operations, to be executed sequentially.

4.14.1 Fields

Field	**Type**	**Description**
name	non-empty string	Name of the operation. This is the key field and a required attribute.
description	string	Free format description.
category	normalizedString	Free format category.
subcategory	normalizedString	Free format subcategory.
location	location	Location of the operation. Default is null. The working hours and holidays for the operation are taken from the ‘available’ calendar of the location.
owner	operation	Operations can be organized in a hierarchical tree. This field defines the parent operation.
fence	duration	Time window from the current date of the plan during which all operationplans are expected to be frozen / released. When the "FENCE" constraint is enabled in the solver, it won’t create any new operation plans in this time fence. Only the externally supplied operationplans will then exist in this time window.
size_minimum	Positive double	A minimum size for operationplans. A request for a lower quantity will be rounded up.
size_multiple	Positive double	A lotsize quantity for operationplans.

cost	double	The cost of executing this operation, per unit of the operation_plan. Depending on what the operation models, this represents transportation costs, manufacturing costs, material procurement costs, delivery costs, etc... The default value is 1.0.
pretime	duration	A pre-operation time, used as a buffer for uncertain material supply. The solver will try to position material supply for operation plans early by the time specified here. This is a soft constraint, ie it can be violated if required to meet the demand in time.
posttime	duration	A post-operation time, used as a buffer for uncertain capacity or operation duration. The solver will try to respect this time as a soft constraint. Ie when required to meet demand on time the post-operation time can be violated. This field is used to model time-based safety stock targets. It is typically set for the producing operation of a certain buffer. If you want to model a safety stock quantity, you can use the minimum field on the buffer.
detectproblems	boolean	Set this field to false to skip problem detection on this operation. The default value is true.
loads	List of load	A list of all resources loaded by this operation.
flows	List of flow	A list of all buffers where material is consumed from or produced into.
level	integer	Indication of how upstream/downstream this entity is situated in the supply chain. Lower numbers indicate the entity is close to the end item, while a high number will be shown for components nested deep in a bill of material. The field is export-only.
cluster	integer	The network of entities can be partitioned in completely independent parts. This field gives the index for the partition this entity belongs to. The field is export-only.
action	A C AC (default) R	Type of action to be executed: • A: Add an new entity, and report an error if the entity already exists. • C: Change an existing entity, and report an error if the entity doesn't exist yet. • AC: Change an entity or create a new one if it doesn't exist yet. • R: Remove an entity, and report an error if the entity doesn't exist.

4.14.2 operation_fixed_time

Models an operation with a fixed duration regardless of the quantity.
E.g. a transport operation.

This is the default operation type.

Field	Type	Description
duration	duration	Duration of the operation. The default value is 0.

4.14.3 operation_time_per

Models an operation where the duration changes linear with the quantity.
E.g. a production operation.

The total duration of the operation plan is the sum of:

- A fixed DURATION.
- A variable duration, computed as the operationplan quantity multiplied by a DURATION_PER.

Field	Type	Description
duration	duration	Fixed component of the duration of the operationplan. The default value is 0.
duration_per	duration	Variable component of the duration of the operationplan. The default value is 0.

4.14.4 operation_alternate

Models a choice between different operations.
It has a list of alternate sub-operations listed, each with a priority.

Field	Type	Description
alternates	List of alternate	List of alternate sub-operations, each with their priority.

Alternate fields:

Field	Type	**Description**(:table border=1 width="100%":)

operation	operation	Sub-operation.
priority	integer	Priority of this alternate. Lower numbers indicate higher priority. When the priority is equal to 0, this alternate is considered unavailable and it can't be used for planning.
effective_start	dateTime	Earliest allowed start date for using this alternate.
effective_end	dateTime	Latest allowed end date for using this alternate.

4.14.5 operation_routing

Models a sequence a number of 'step' sub-operations, to be executed sequentially.

Field	Type	Description
steps	List of operation	Lists all sub-operations in the order of execution.

4.14.6 Example XML structures:

- Adding or changing operations

```
<plan>
  <operations>
    <operation name="buy item X from supplier"
      xsi:type="operation_fixed_time">
      <duration>P1D</duration>
    </operation>
    <operation name="make item X"
      xsi:type="operation_time_per">
      <duration>PT1H</duration>
      <duration_per>PT5M</duration_per>
    </operation>
    <operation name="make or buy item X"
      xsi:type="operation_alternate">
      <alternates>
        <alternate>
          <operation name="make item X" />
          <priority>1</priority>
        </alternate>
        <alternate>
          <operation name="buy item X from supplier" />
          <priority>2</priority>
        </alternate>
```

```
      </alternates>
    </operation>
    <operation name="make subassembly"
      xsi:type="operation_routing">
      <steps>
         <operation name="make subassembly step 1"
           duration="PT1H"/>
         <operation name="make subassembly step 2"
           duration="PT5M"/>
      </steps>
    </operation>
  </operations>
</plan>
```

- Deleting an operation

```
<plan>
   <operations>
      <operation name="make item X" action="R"/>
   </operations>
</plan>
```

4.14.7 Example Python code:

- Adding or changing operations

```
op1 = frepple.operation_fixed_time(name="buy item X from supplier",
       duration=24*3600)
op2 = frepple.operation_time_per(name="make item X",
       duration=3600, duration_per=60*5)
op3 = frepple.operation_alternate(name="make or buy item X")
op4 = frepple.operation_routing(name="make subassembly")
```

- Deleting an operation

```
frepple.operation(name="make item X", action="R")
```

- Iterate over operations, loads and flows

```
for o in frepple.operations():
  print "Operation:", o.name, o.description, o.category
  for l in o.loads:
    print "  Load:", l.resource.name, l.quantity,
      l.effective_start, l.effective_end
  for l in o.flows:
    print "  Flow:", l.buffer.name, l.quantity,
      l.effective_start, l.effective_end
```

4.15 OperationPlan

Used to model an existing or planned activity.
This can represent work-in-progress, in-transit shipments, planned material receipts, frozen manufacturing plans, etc…

4.15.1 Fields

Field	Type	Description
operation	non-empty string	Name of the operation. This field is required when no identifier is provided.
id	unsignedLong	Unique identifier of the operationplan. If left unspecified an identifier will be automatically generated. This field is required when updating existing instances.
start	dateTime	Start date.
end	dateTime	End date.
demand	demand	Points to the demand being satisfied with this operationplan. This field is only non-null for the actual delivery operationplans.
quantity	double	Quantity being planned.
locked	boolean	A locked operation plan is not allowed to be changed any more by any solver algorithm.
owner	operation_plan	Points to a parent operationplan. The default is NULL.
action	A C AC (default) R	Type of action to be executed: • A: Add an new entity, and report an error if the entity already exists. • C: Change an existing entity, and report an error if the entity doesn't exist yet. • AC: Change an entity or create a new one if it doesn't exist yet. • R: Remove an entity, and report an error if the entity doesn't exist.

4.15.2 Example XML structures:

- Adding an operationplan to represent a planned receipt of material

```
<plan>
  <operationplans>
    <operationplan operation="Purchase component A">
```

```
      <quantity>100</quantity>
      <start>2007-01-10T00:00:00</start>
      <locked>true</locked>
    </operationplan>
  </operationplans>
</plan>
```

- Deleting an operationplan

```
<plan>
   <operationplans>
      <operationplan id="1020" action="R"/>
   </operationplans>
</plan>
```

4.15.3 Example Python code:

- Adding an operationplan to represent a planned receipt of material

```
op = frepple.operation(name="Purchase component A", action="C")
opplan = frepple.operationplan(operation=op,
   quantity=100, start=datetime.datetime(2007,1,10), locked=True)
```

- Deleting an operationplan

```
frepple.operationplan(id="1020",action="R&quot)
```

- Iterate over operationplans

```
for i in frepple.operationplans():
  print i.operation.name, i.quantity, i.start, i.end
```

4.16 Problem

FrePPLe will automatically detect problems and inconsistencies in the plan.

Problem detection can optionally be disabled on entities by setting the field "DETECTPROBLEMS" to false.

Problems are export-only, i.e. you can't read them as input.

4.16.1 Types

Problem Entity	Problem Category	Description
demand	unplanned	No plan exists yet to satisfy this demand.
demand	excess	A demand is planned for more than the requested quantity.
demand	short	A demand is planned for less than the requested quantity.

demand	late	A demand is satisfied later than the accepted tolerance after its due date
demand	early	A demand is planned earlier than the accepted tolerance before its due date.
resource	overload	A resource is being overloaded during a certain period of time.
resource	underload	A resource is loaded below its minimum during a certain period of time.
buffer	material excess	A buffer is carrying too much material during a certain period of time.
buffer	material shortage	A buffer is having a material shortage during a certain period of time.
operationplan	before current	Flagged when an operationplan is being planned in the past, i.e. it starts before the current date of the plan.
operationplan	before fence	Flagged when an operationplan is being planned before its fence date, i.e. it starts 1) before the current date of the plan plus the release fence of the operation and 2) after the current date of the plan.
operationplan	precedence	Flagged when the sequence of two operationplans in a routing isn't respected.

4.16.2 Fields

Field	Type	Description
name	normalizedString	Problem type.
description	normalizedString	Description of the problem.
weight	double	A number expressing the seriousness of the problem.
start	dateTime	Date at which the problem starts.
end	dateTime	Date at which the problem ends.

4.16.3 Example Python code:

- Iterate over all problems

```
for i in frepple.problems():
   print i.entity, i.name, i.description, i.start, i.end, i.weight
```

4.17 Resource

Resources represent capacity.
They represent a machine, a worker or a group of workers, or some logical limits.

A calendar refers to a time-phased maximum limit of the resource usage.

Operations will consume capacity using loads.

Different types of resources exist:

- resource_default (p 64):
  A default resource is constrained with a maximum available capacity.
- resource_infinite (p 64):
  An infinite resource has no capacity limit.

4.17.1 Fields

Field	**Type**	**Description**
name	non-empty string	Name of the resource. This is the key field and a required attribute.
description	string	Free format description.
category	normalizedString	Free format category.
subcategory	normalizedString	Free format subcategory.
owner	resource	Resources can be organized in a hierarchical tree. This field defines the parent resource. No specific planning behavior is currently linked to such a hierarchy.
members	list of resource	Resources can be organized in a hierarchical tree. This field defines a list of child resources.
location	location	Location of the resource. Default is null. The working hours and holidays for the resourceare taken from the ‘available’ calendar of the location.
maximum	calendar	Refers to a calendar storing the available capacity. A problem is reported when the resource load exceeeds than this limit.
cost	double	The cost of using 1 unit of this resource for 1 hour. The default value is 1.0.
detectproblems	boolean	Set this field to false to supress problem detection on this resource. Default is true.

loads	list of load	Defines the capacity of the operations.
loadplans	list of loadplan	This field is populated during an export with the plan results for this resource. It shows all the resource load profile. The field is export-only.
level	integer	Indication of how upstream/downstream this entity is situated in the supply chain. Lower numbers indicate the entity is close to the end item, while a high number will be shown for components nested deep in a bill of material. The field is export-only.
cluster	integer	The network of entities can be partitioned in completely independent parts. This field gives the index for the partition this entity belongs to. The field is export-only.
action	A C AC (default) R	Type of action to be executed: • A: Add an new entity, and report an error if the entity already exists. • C: Change an existing entity, and report an error if the entity doesn't exist yet. • AC: Change an entity or create a new one if it doesn't exist yet. • R: Remove an entity, and report an error if the entity doesn't exist.

4.17.2 resource_default

A default resource is constrained with a maximum available capacity.

No fields are defined in addition to the ones listed above.

4.17.3 resource_infinite

An infinite resource has no capacity limit.
It is useful to monitor the loading or usage.

The MAXIMUM field is unused for this resource type.

4.17.4 Example XML structures:

- Adding or changing a resource

```
<plan>
  <resources>
    <resource name="machine X">
      <maximum name="capacity calendar for machine X" />
```

```
    </resource>
  </resources>
</plan>
```

- Deleting a resource

```
<plan>
   <resources>
      <resource name="machine X" action="R"/>
   </resources>
</plan>
```

4.17.5 Example Python code:

- Adding or changing a resource

```
cal = frepple.calendar(name="capacity calendar for machine X")
res = frepple.resource(name="machine X", maximum=cal)
```

- Deleting a resource

```
frepple.resource(name="machine X", action="R")
```

- Iterater over resources, loads and loadplans

```
for r in frepple.resources():
  print "Resource:", r.name, r.description, r.category
  for l in r.loads:
    print "  Load:", l.operation.name, l.quantity,
      l.effective_start, l.effective_end
  for l in r.loadplans:
    print "  Loadplan:", l.operationplan.operation.name,
      l.quantity, l.startdate, l.enddate, l.operationplan.id
```

4.18 Solver

A solver represents modules of functionality that manipulate the model.
Examples are solvers to generate a plan, solvers to compute safety stocks, solvers to create production or purchase orders, etc…

Only one solver is included in the core library: solver_mrp (p 66), which uses a heuristic algorithm to generate plans.
Other solvers are implemented as optional modules.

For running a solver see the command command_solve.

4.18.1 Fields

Field	Type	Description

name	non-empty string	Name of the solver. This is the key field and a required attribute.
loglevel	0 - 3	Amount of logging and debugging messages: • 0: Silent operation. Default logging level. • 1: Show minimum output. • 2: Show standard output. • 3: Show debugging output.
action	A C AC (default) R	Type of action to be executed: • A: Add an new entity, and report an error if the entity already exists. • C: Change an existing entity, and report an error if the entity doesn't exist yet. • AC: Change an entity or create a new one if it doesn't exist yet. • R: Remove an entity, and report an error if the entity doesn't exist.

4.18.2 solver_mrp

Field	Type	Description
constraints	unsignedShort	Sum up the values of the constraints you want to enable in the solver: • 1: Lead times, ie don't plan in the past • 2: Material supply, ie don't allow inventory values to go negative • 4: Capacity, ie don't allow to overload resources • 8: Operation fences, ie don't allow to create plans in the frozen fence of operations
maxparallel	Positive integer	Specifies the number of parallel threads the solver creates during planning. The default value depends on whether the solver is run in verbose mode or not: • In normal mode the solver uses as many threads as specified by the NUMBER_OF_PROCESSORS environment variable. • When the logging level is different from 0 the solver runs in a single thread to avoid mangling the debugging output. of different threads.

4.18.3 Example XML structures:

- Adding or changing a solver

```
<plan>
   <solvers>
      <solver name="MRP" xsi-type="solver_mrp">
        <constraints>7</constraints>
        <maxparallel>2</maxparallel>
      </solver>
   </solvers>
</plan>
```

- Deleting a solver

```
<plan>
   <solvers>
      <solver name="MRP" action="R"/>
   </solvers>
</plan>
```

4.18.4 Example Python code:

- Adding or changing a solver, and running it

```
sol = frepple.solver_mrp(name="MRP", constraints=7, maxparallel=2)
sol.solve()
```

- Deleting a solver

```
frepple.solver(name="MRP", action="R")
```

CHAPTER

5

Solver algorithm

Different solvers and algorithms can be used with the frePPle models.

FrePPLe comes with a default solver that is documented in this chapter.
It is based on a heuristic algorithm, structured in a clear ask-reply pattern between the different entities.

The algorithm can create different types of plans. With the following three flags, a total of 8 combinations are possible:

- Material constrained or not:
  Supply of raw material can be treated as finite or infinite.
- Capacity constrained or not:
  Production capacity can be treated as finite or infinite.
- Leadtime constrained or not:
  Allow or disallow plans to be created in the past.

It is possible to build create extensions to the solver, or to create a completely new solver altogether. The solvers can be loaded as plugin modules without touching or recompiling the main application.

5.1 Solver features

In brief, here are the main features of the solver:

5.1.1 Solver

- Ability to create **unconstrained plans**.
- Ability to respect following **constraints: material supply, available capacity, leadtime, release time fence**.
- Ability to run in **multi-threaded** mode. Different threads are solving independent sub-problems.

5.1.2 Demand

- **Demand priorities** are recognized, such that constraints impact the lowest ranking demands only.
  The default ranking is based on the priority attribute and the due date.
- Ability to respect different **demand policies**: In case of a constraint a demand can be allowed to be satisfied late or not. Satisfying the demand in multiple parts can be allowed or not.

5.1.3 Operation

- Models **multiple operation types**.
  - Operations with fixed duration.
  - Operations with variable duration, depending on quantity.
  - Alternate operations: When a demand can't be met from the primary operation the solver will plan on alternative operations.
  - Date-effective operations: Depending on the start date (or end date) different operations are used.
  - Multi-step operations: An operation can have multiple sub-operations that need to be executed in sequence.
- The operations can be planned as a multiple of the **lot-size** quantity.
- A **minimum size** can be enforced when planning an operation.
- **Pre- and post-operation times** used as soft constraints (ie they are respected when feasible but will be reduced when required to meet the demand in time).

5.1.4 Resource

- Resources loaded during the complete duration of an operation.
- Resources with **finite or infinite capacity**.
- Capacity shortages are solved by **moving operations early**.

5.1.5 Buffer

- Material consumption or production happens at the start or at the end of operations.
- Buffers with **finite or infinite material supply**.

- Ability to specify a desired minimum inventory level, aka **safety stock**. The minimum level can be time dependent and is treated as a soft constraint (ie will be respected when feasible, but will be violated when constraints prevent meeting it).

5.2 Implementation details

The algorithm solves demand per demand. The demand is thus sorted in descending order of priority, and next these demands are planned one after the other.

When planning a single demand, the algorithm basically consists of a set of recursive functions structured in a ask-reply pattern, as illustrated in the example below. The indention is such that the ask and its matching reply are represented at the same level.

> Every demand has a certain delivery operation associated with it ,either directly or indirectly by specifying a delivery operation for the requested item. The demand **asks** this **operation** for the requested quantity on the due date of the demand.
>
> > (*) The operation first checks for the lead time constraints.
> >
> > The operation will **ask** each of the **loads** to verify the capacity availability.
> >
> > The operation will **ask** each of the **flows** to check the availability of consumed materials.
> >
> > > A load passes on the question and **asks** the **resource**.
> > >
> > > > The **resource reply** indicates whether the capacity is available or not.
> > >
> > > The **load** uses the resource reply to **reply** to the operation.
> > >
> > > A flow passes on the question too and **asks** the **buffer**.
> > >
> > > > The buffer checks the inventory situation.
> > > >
> > > > If material is available no further recursion is required.
> > > >
> > > > If the required material isn't available the buffer will **ask** an **operation** for a new replenishment. Each buffer has a field indicating which operation is to be used to generate replenishments.
> > > >
> > > > Depending on the buffer inventory profile, safety stock requirements, etc... the operation may be asked for different quantities and on different dates than the original demand.
> > > >
> > > > > When an operation is asked to generate a replenishment it evaluates the leadtime, material and capacity constraints. This results in a nested ask-sequence similar as the one described earlier - marked with (*)
> > > > >
> > > > > ...

The maximum recursion depth will be the same as the number of levels in the bill-of-material of the end item.

In some cases the iteration can be stopped at an intermediate level. Eg. When sufficient inventory is found in a buffer and no replenishment needs to be asked: a positive reply can be returned immediately.

Eg. When an operation would need to be planned in the past (ie leadtime constraint violated) a negative reply can be returned immediately.

...

The operation collects the replies from all its flows, loads and -indirectly- from all entities nested at the deeper recursion levels. A final **reply** of the **operation** is generated.

Based on the reply of the replenishing operation the **buffer** evaluates whether or not the replenishments are possible, and **replies** back to the flow. Sometimes a buffer may need to ask multiple times for a replenishment before an answer can be returned.

The **flow** picks up the buffer reply and **replies** to the operation.

From the reply of all its loads and flows the **operation** compiles a **reply** and returns it to the demand. The interaction of material, leadtime and capacity constraint are pretty complex and an operation may require several ask-reply iterations over its flows and loads before a final answer can be returned.

The answer of the operation indicates how much of the requested quantity can be satisfied on the requested date.

Depending on the planning result and the demand parameters (such as allow/disallow satisfying the demand late or in multiple deliveries) we can now decide to commit all operation plans created during the whole ask-reply sequence.

If we're not happy with the reply the operation plans created are undone again and we can go back to the first step and ask for the remaining material or at a later date.

The answer in each of the above steps consists of 1) ask-quantity and 2) ask-date.
The reply used in each of the above steps consists of 1) reply-quantity and 2) reply-date. The reply-quantity represents how much of the requested quantity can be made available at the requested date. The reply-date is useful when the ask can not -or only partially- be met: it then indicates the earliest date when the missing quantity might be possible.

In the above sequence the steps are described at a very high level.
In the following sections each of the different ask-reply steps are now explained in further detail.

1. Top level loop
2. Demand solver
3. Buffer solver
4. Operation solver
5. Flow solver
6. Load solver
7. Resource solver

5.2.1 Top level loop

Delete the existing operation-plans, as far as they aren't locked.
Identify the clusters to be planned.
Categorize the demand to be planned by cluster and sort them by priority.
Create parallel threads for the planning.
In each planning thread, loop through all demands.

Call demand→solve()

5.2.2 Demand solver

Ignore the demand if quantity is 0
Erase previous delivery operation plans, except the ones that are locked
Loop until the full demand quantity is planned.

Call operation→ask(missing quantity,due date), where operation is the demand's or the items delivery operation

If planned quantity = requested quantity, or the demand planning policy allows planning the demand in parts or shorts then

Commit the operation plan creation

Else

Clear the list of scheduled operation plans

If planned quantity > 0 then

// This last step is required to make sure all supplying paths are planned for the quantity of the most constraining path

Call operation→ask(planned quantity, due date)

Commit the operation plan creation

Update the planned quantity for the next iteration in the loop

Exit the loop if the demand can't be planned late

5.2.3 Buffer solver

Standard buffer

Buffer is asked for a quantity Q at the date D For each flowplan on the buffer

If the on-hand value is positive

Set the variable ExtraInventoryDate if it is not set before. This variable stores the date when there is additional, unallocated inventory available.

Else if the on-hand value is negative

Compute the shortage as current onhand required minimum quantity + known shortage from previous dates

If a producing operation exists

Try to get extra supply for the shorted quantity. This replenishment will update the onhand value of the current flowplan

If the onhand is still less than the required minimum quantity - the known shortage

This situation happens when the producing operation can't replenish the buffer enough, or when all supply in a buffer without producing operation has been exhausted.

Increase the variable storing the known shortage at previous dates.

Reset the ExtraInventoryDate if it was set.

If there is a shortage, a producing operation exists and the above loop didn't already do the following

Try to get more supply at the requested date.

Not only can this reduce the shortage, but also important is the next-date returned by the producing operation.

Note that if this step creates more supply to meet the demand, that supply is not positioned such that inventory is minimized. The flowplan loop does minimize the inventory by replenishing only when the inventory drops below the minimum.

The final results are now:

Returned quantity: requested quantity shortages

Returned date:

= requested date if there is no shortage

Or = reply date of the producing operation

Or = ExtraInventoryDate if that is less than the operation reply date

todo Not up to date with the pre-op time loop…

Infinite buffer

Always reply for the full quantity.

5.2.4 Operation solver

Fixed time and time-per operation

Operation is asked for a quantity Q at the date D
Create required operation plan descriptor
Loop backward in time D until we have have found the full recource capacity
Call Operation→ask(Qremaining, Dupdated)
For each consuming flow

Ask the buffer for the planned quantity on the requested date

Update Qremaining and Dupdated
Return the accumlated promise quantity

@todo incomplete documentation: need description of leadtime constraints + flowplan call + loadplan call

Alternate operation

Operation is asked for a quantity Q at the date D
Remaining quantity = Q
Next ask date = infinite future
Loop through all alternate sub operations

Create top operation plan descriptor

Call Operation→ask(Remaining quantity, D)

If some quantity could be planned along the alternate

Check for material and capacity constraints on the top operation plan

Reduce the remaining quantity

Break out of the loop if the requested quantity is completely planned

Else

If the next ask date of the alternate is less than the current minimum, update the next ask date

Return the planned quantity and the next ask date

Routing operation

Operation is asked for a quantity Q at the date D
Create the top operation plan

Check the flowplans and loadplans of the top operation plan
Initialize Q2 to Q and D2 to D
For all steps of the routing

Call operation→ask(Q2,D2)

Update Q2 if planned quantity < Q2

Update D2 with the operation time

5.2.5 Flow solver

If the requested date is outside of the effective date range of the flow, reply for the full requested quantity.
(@todo this date range isn't implemented yet in the flow model, and the check isn't implemented yet)
Otherwise, ask the buffer to generate the reply for the quantity and date.

5.2.6 Load solver

If the requested date is outside of the effective date range of the load, reply for the full requested quantity.
(@todo this date range isn't implemented yet in the load model, and the check isn't implemented yet)
Otherwise, ask the resource to generate the reply for the quantity and date.

5.2.7 Resource solver

Standard resource

The sequence below show the interaction between the functions checkOperationCapacity(OperationPlan*), Solve(Load*) and Solve(Resource*).

An operationplan is asked to check for capacity problems (not for a date & quantity)
Loop through all loadplans of the operationplan

Call the load solver

If this is not an ending loadplan or it has a zero quantity, move on to the next loadplan

Call the resource solver

// Look if the operationplan overloads the resource

Set HasOverload to false. (*)

Start recursing backwards in the timeline starting from the ending loadplan

While HasOverload is still false and not yet at the very start

If the resource loading > maximum

Break out of the while loop

// Solve any overloads by reducing the operationplan quantity

If HasOverload and there is a period where the resource isn't overloaded yet

Resize the operationplan to fit in this time window

If the resizing is successful

There is no longer an overload problem

Set HasOverload to false

Else

Restore the original time and quantity of the operationplan

// Solve any overloads by using earlier capacity

If HasOverload

Search going back in time till the resource loading < maximum

If available capacity was found

Move the operation plan to end at that time in the timeline

Go back to the step marked with (*)

Else

Reply quantity will be zero: No available capacity was found

// Look for overloads, and try to solve them using later capacity

If the reply quantity is 0

Find the date after the ask date where the load drops below the maximum (**)

Move the operationplan such that it starts at this date

If the operationplan still overloads the resource

Go back to step (**) and try another, later date

Else

Reply quantity is 0 and the reply next-date is the end date of the moved operationplan

If in the above loop the operation plan is moved to a new date, the complete loop over all loadplans must be repeated.

Infinite resource

The loop is similar to the above, except that the resource solver will always reply an okay.

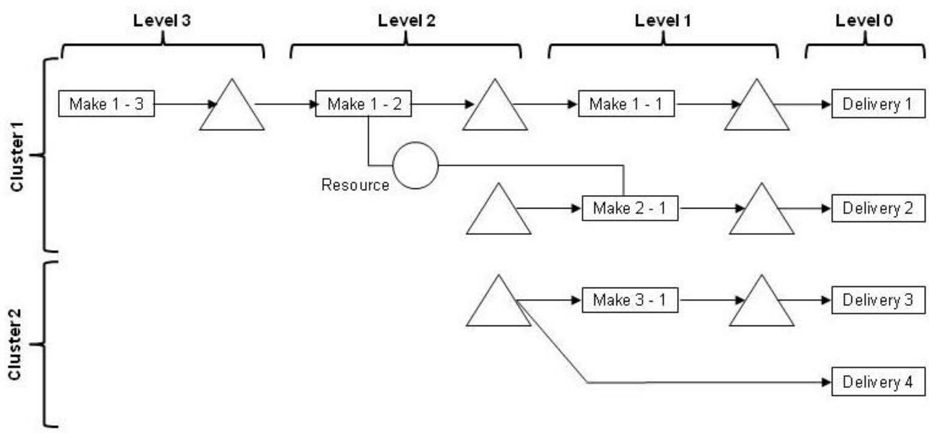
5.3 Cluster and level algorithm

Resources, operations and buffers are connected with each other with loads and flows. An operation has a collection of loads and flows. Each flow establishes a connection with a buffer, and each load a connection with a resources. The entities thus constitute a network graph. In this network context we define clusters and level as follows.

A **cluster** is a set of connected entities. When a network path across loads and flows exists between 2 entities they belong to the same cluster. When no such path exists they are effectively situated in independent sub-networks and clusters.
Internally, each cluster is represented by a number.
Clusters allow us to group entities and are very useful in multithreaded environment: since the clusters are completely independent we can use different threads to solve each cluster as a separate subproblem.

Material flows in the network have a direction. This creates a sense of direction in our network which is expressed by the **level** concept.
An operation consumes and produces material, as defined by the flow entities (aka bill of material or recipe).
In this context the level is a number that is defined such that the level of a consumed material buffer is always higher than the level of the produced material buffer. The demand is normally (but not exclusively!) placed on the material buffers with level 0, and the level number increases as we recurse through the different levels in the bill of material.
Raw materials have the highest level number.

The level and cluster number are helpful for the various solver algorithms. They provide valuable information about the structure of the network.



The algorithm used to compute the level and cluster information is based on a walk through the network: We select an unmarked operation and recurse through the loads and flows to find all connected entities, updating the cluster and level information as we progress.

For efficiency, the algorithm is implemented as a lazy function, i.e. the information is only computed when the user is retrieving the value of a level or cluster field. The algorithm is not incremental (yet), but computes the information for the complete network in a single pass: a change to a single entity will trigger re-computation of all level and cluster information for all entities.

Note: An updated algorithm has been designed for the cluster computation. Its advantage compared to the current implementation is a much better effiency in the case of frequent model updates. The computation will be completely incremental, compared to the single pass for all entities in the current implementation.

The detailed flow of the algorithm is as follows:

// Initialisation
Lock the function
Reset the level and cluster to -1 on all resources, operations and buffers
Reset the total number of clusters

// Main loop
Loop through all operations

If the operation has no producing flow

Activate the level computation

If the operation isn't part of a cluster yet

Activate the cluster computation

Increment the cluster counter

If both cluster and level computation are inactive, move on to the next operation

Push the current operation on the recursion stack, with level 0 or -1

Loop until the stack is empty

Pop an operation from the recursion stack

Pop the value of cur_level from the stack

Loop through the sub operations and super operations

If their level is less than the current level

Push sub operation on the stack, with the same level as the current operation

Set the level and cluster fields

Else if cluster is not set yet

Push sub operation on the stack, with -1 as the level

Set the cluster field

Loop through all loadplans of the operation

If level search is active and the resource level is less than the level of the current operation

Update the level of the resource

If the cluster of the resource is not set yet

Set the cluster of the resource

Loop through all operations that are loading the resource

If operation cluster isn't set yet

Push the operation on the stack, level -1

Set the cluster of the operation

Loop through all flows of the current operation

If this is a consuming flow and level_search is active and the level of the buffer is less than the current level +1

Level recursion is required

If level recursion is required or the cluster of the buffer is not set yet

Set the cluster of the buffer

Loop through all flows connected to the buffer

If it is a consuming flow and level search recursion was enabled

todo incomplete documentation

// Catch buffers missed by the main loop
Loop through all buffers which don't have any flow at all.

Increment the total number of clusters

Set the cluster number to the new cluster

// Catch resources missed by the main loop
Loop through all resources which don't have any load at all.

Increment the total number of clusters

Set the cluster number to the new cluster

// Finalization
Unlock the function

CHAPTER

6

Extension modules

FrePPLe can easily be extended with modules that are loaded at runtime.
This chapter describes the modules that are provided with frePPLe.

To load an extra module, you need to update the following 2 files in the FREPPLE_HOME directory:

- Add a loadlib command in the file **init.xml**. This file is automatically executed when frePPLe starts.
- Edit the file **frepple.xsd** to include an additional XML schema file. The new file defines the new XML data types that are enabled by the new module.

The default version of these files enables the forecast module only.

The C++ code required to create a custom module is described in the developer section of this manual: Extension modules. An example is also availabe in the Test Sample Module

1. Forecast module
2. Webservice module
3. Linear programming solver module

6.1 Forecast module

The forecast module provides the following functionality for representing forecasted future demand:

- A **new demand type** to model forecasts.
  A calendar model is used to divide the time horizon into a number of time buckets. A demand is automatically created for each time bucket.
  See the example below.
- Functionality for **distributing / profiling** forecast numbers into time buckets used for planning.
  This functionality allows to translate between different time granularities.

The forecast entered by the sales department could for instance be in monthly buckets, while the manufacturing department requires the forecast to be in weekly or even daily buckets to generate accurate manufacturing and procurement plans.
Another usage is to model a delivery date profile of the customers. Each bucket has a weight that is used to model situations where the demand is not evenly spread across buckets: e.g. when more orders are expected due on a monday than on a friday, or when a peak of orders is expected for delivery near the end of a month.

- A solver for **netting orders from the forecast**.
  As customer orders are being received they need to be deducted from the forecast to avoid double-counting it. The netting solver will for each order search for a matching forecast and reduce the remaining net quantity of the forecast.
  For example, assume the forecast for customer A in January is 100 pieces, and we have already received orders of 20 from the customer.
  Without the netting algorithm the demand in January will be 120 pieces, which is (very likely) not correct.
  The netting solver will deduct the orders of 20 from the forecast. The total demand that is planned in January will then be equal to 100: 80 remaining forecast + 20 orders.
  The solver algorithm has logic to match a demand with the most appropriate forecast, and can also consider netting in previous and subsequent time buckets.
- A forecasting algorithm to **extrapolate historical demand data to the future**.
  The following classical forecasting methods are implemented:
  - single exponential smoothing, which is applicable for constant demands
  - double exponential smoothing, which is applicable for trended demands
  - moving average, which is applicable when there is little demand history to rely on

  The forecast method giving the smallest mean absolute deviation (aka "mad"-error) will be automatically picked to produce the forecast.
  The algorithm will automatically tune the parameters for the forecasting methods (i.e. alfa for the single exponential smoothing, or alfa and gamma for the double exponential smoothing) to their optimal value. The user can specify minimum and maximum bounaries for the parameters and the maximum allowed number of iterations for the algorithm.

The module enables the following new objects:

- demand_forecast (p 83) is a specialized representation of the demand.
- solver_forecast (p 83) is a solver for performing the forecast netting calculations.

6.1.1 Module configuration

The module support the following configuration parameters:

- **Net.CustomerThenItemHierarchy**:
  As part of the forecast netting a demand is assiociated with a certain forecast. When no matching forecast is found for the customer and item of the demand, frePPLe looks for forecast at higher level customers and items.
  This flag allows us to control whether we first search the customer hierarchy and then the item hierarchy, or the other way around.
  The default value is true, ie search higher customer levels before searching higher levels of the item.

- **Net.MatchUsingDeliveryOperation**:
  Specifies whether or not a demand and a forecast require to have the same delivery operation to be a match.
  The default value is true.
- **Net.NetEarly**:
  Defines how much time before the due date of an order we are allowed to search for a forecast bucket to net from.
  The default value is 0, meaning that we can net only from the bucket where the demand is due.
- **Net.NetLate**:
  Defines how much time after the due date of an order we are allowed to search for a forecast bucket to net from.
  The default value is 0, meaning that we can net only from the bucket where the demand is due.
- **Forecast.Iterations**:
  Specifies the maximum number of iterations allowed for a forecast method to tune its parameters.
  Only positive values are allowed and the default value is 10.
  Set the parameter to 1 to disable the tuning and generate a forecast based on the user-supplied parameters.
- **Forecast.madAlfa**:
  Specifies how the MAD forecast error is weighted for different time buckets. The MAD value in the most recent bucket is 1.0, and the weight decreases exponentially for earlier buckets.
  Acceptable values are in the interval 0.5 and 1.0, and the default is 0.95.
- **Forecast.Skip**:
  Specifies the number of time series values used to initialize the forecasting method. The forecast error in these bucket isn't counted.
- **Forecast.MovingAverage.buckets**
  This parameter controls the number of buckets to be averaged by the moving average forecast method.
- **Forecast.SingleExponential.initialAlfa**,
  Forecast.SingleExponential.minAlfa,
  Forecast.SingleExponential.maxAlfa:
  Specifies the initial value and the allowed range of the smoothing parameter in the single exponential forecasting method.
  The allowed range is between 0 and 1. Values lower than about 0.05 are not advisible.
- **Forecast.DoubleExponential.initialAlfa**,
  Forecast.DoubleExponential.minAlfa,
  Forecast.DoubleExponential.maxAlfa:
  Specifies the initial value and the allowed range of the smoothing parameter in the double exponential forecasting method.
  The allowed range is between 0 and 1. Values lower than about 0.05 are not advisible.
- **Forecast.DoubleExponential.initialGamma**,
  Forecast.DoubleExponential.minGamma,
  Forecast.DoubleExponential.maxGamma:
  Specifies the initial value and the allowed range of the trend smoothing parameter in the double exponential forecasting method.
  The allowed range is between 0 and 1.

6.1.2 Demand subclass demand_forecast

All fields available on the demand model are allowed on a forecast.
In particular the item and customer field are important, since these are used to match a demand with a certain forecast for netting.

The following fields are available in addition to the demand fields.

Field	Type	Description
calendar	non-empty string	Name of the calendar used to define time buckets for distributing the forecast numbers.
discrete	boolean	Specifies whether forecast should be rounded to integer numbers. The default value is true.
bucket buckets	xml	Speficies the forecast value for a date range. See example below.

6.1.3 Solver solver_forecast

This solver runs the forecast netting calculations.
The solver loops through the demands in order of their priority, and for each demand a matching forecast is searched. When a matching forecast is identified, the solver looks for a time bucket to net from: first in the bucket where it is due, then in earlier buckets within the chosen time window, and finally in later buckets within the chosen time window. The net forecast in the forecast buckets is decreased.

Note the the profiling of the forecast is not handled by this solver. The profiling happens during the data load, i.e. when the forecast demand is read in.

In addition to the regular solver fields, the following fields are available.

Field	Type	Description
loglevel	0 - 2	Amount of logging and debugging messages: • 0: Silent operation. Default logging level. • 1: Log demands being netted and the matching forecast. • 2: Same as 1, plus details on forecast buckets being netted.

6.1.4 Example XML structures:

- Loading the module

```
<plan>
  <commands>
    <command xsi:type="command_loadlib" filename="mod_forecast.so">
      <parameter name="Net.CustomerThenItemHierarchy" value="true" />
      <parameter name="Net.MatchUsingDeliveryOperation" value="true" />
      <parameter name="Net.NetEarly" value="P7D" />
      <parameter name="Net.NetLate" value="P7D" />
    </command>
  </commands>
</plan>
```

- Forecast input

```
<plan>
  <demands>
     <demand name="Forecast 1" xsi:type="demand_forecast">
       <item name="Product 1" />
       <customer name="Customer 1" />
       <calendar name="planningbuckets" />
       <discrete>true</discrete>
       <buckets>
         <bucket>
           <start>Monday, 1 January 2007T00:00:00</start>
           <end>Thursday, 1 February 2007T00:00:00</end>
           <total>200</total>
         </bucket>
         <bucket>
           <start>Thursday, 1 February 2007T00:00:00</start>
           <end>Thursday, 1 March 2007T00:00:00</end>
           <total>200</total>
         </bucket>
       </buckets>
     </demand>
  </demands>
</plan>
```

- Netting customer orders from the forecast

```
<plan>
  <commands>
     <command xsi:type="command_solve">
       <solver name="Netting" xsi:type="solver_forecast">
         <loglevel>1</loglevel>
       </solver>
     </command>
  </commands>
</plan>
```

6.1.5 Example Python code:

- Adding or changing a forecast

```
it = frepple.item(name="item")
cust = frepple.customer(name="customer")
cal = frepple.calendar(name="planningbuckets")
fcst = frepple.demand_forecast(name="My forecast",
  item=it, customer=cust, calendar=cal)
```

- Creating a time series forecast
  The first argument is the demand history in previous buckets.
  The second argument are the time buckets where we want to create a forecast value.

```
thebuckets = [ i.start for i in thecalendar.buckets ]
fcst.timeseries([10,12,9,11,8,15,19,11], thebuckets)
```

- Netting customer orders from the forecast

```
frepple_forecast.solver_forecast(name="Netting", loglevel=1).solve()
```

6.2 Webservice module

This module implements a multi-threaded SOAP webservice server.
Using the webservice frePPLe can make the plan information on-line accessible to other systems and users, and also receive updated information. In a **Service Oriented Architecture** (SOA) such data exchanges are used to build **composite applications**: data from different services is combined to build rich and flexible applications.

The module is built using the excellent gSOAP toolkit. FrePPLe currently provides only a basic service setup, and doesn't support any of the more advanced gSOAP functionalities, such as HTTPS/SSL, compression, HTTP cookies, SOAP Headers, HTTP basic authentication...
The supported SOAP operations also provide only a limited interface to the frePPLe functionality.

The module enables the following new objects:

- command_webservice (p 85) is a command to run the web service.

6.2.1 Module configuration

The module support the following configuration parameters:

- **port**:
  The port number used by the webservice.
  When left unspecified, the default port number is 6262.
- **threads**:
  Specifies the number of worker threads to create to serve requests.
  The default value is 10.

6.2.2 Command command_webservice

This command runs the multi-threaded webservice. Since the command will wait forever for incoming connections this command should be called as the latest command in the command sequence.

Field	Type	Description
verbose	boolean	When enabled the status of the service is echoed during operation. The default is false.

6.2.3 Example XML structures

- Loading the module:

```
<plan>
  <commands>
    <command xsi:type="command_loadlib" filename="mod_webservice.so">
      <parameter name="port" value="6262" />
      <parameter name="threads" value="10" />
    </command>
  </commands>
</plan>
```

- Running the webservice:

```
<plan>
  <commands>
     <command xsi:type="command_webservice"/>
  </commands>
</plan>
```

6.3 Linear programming solver module

This module implements a linear programming solver.
The solver is intended primarly for prototyping purposes. A linear programming model can quickly be built and validated in a generic way.

Important: This solver module is licensed under the GPL, which is different from the GLPL license normally used by frePPLe.

The module uses the "GNU Linear Programming Kit" library (aka GLPK) to solve the LP model
The solver works as follows:

- The solver expects a **model file** and a **data file** as input.
  The model file represents the mathematical representation of the problem to solve. It can be edited to meet your specfic business problem.
  The data file holds the data to be loaded into the problem. If no data file is specified, the data

section in the model file is used instead.
The user needs to create these files. A convenient way to generate the data file is to use the Python module. See the unit test lp_solver1 for an example.
- The solver solves for a number of objectives in sequence.
After solving an objective's optimal value, the solver freezes the objective value as a constraint and start for the next objective. Subsequent objectives can thus never yield a solution that is suboptimal for the previous objectives.
- After solving for all objectives the solution is written to a **solution file**.
The user is responsible for all processing of this solution file. A convenient way is again to use the Python module.

The unit test lp_solver1 shows how a capacity allocation problem is solved with the module. Different business problems will obviously require a different formulation.

6.3.1 Technical implementation

The module is based on the GLPK (GNU Linear Programming Kit) package. More information on the package can be found on http://www.gnu.org/software/glpk/glpk.html.

Go through the following steps for a typical usage of this solver:

- Load the Python and the LPsolver modules with commands as follows in the init.xml file:

```
<command xsi:type="command_loadlib" filename="mod_python.so" />
<command xsi:type="command_loadlib" filename="mod_lp_solver.so" />
```

- Copy your model file and Python code into your $FREPPLE_HOME directory.
Assume the function exportData is used for exporting the data file, and the function importSolution is used to read the solution file.
- Export the data files, run the solver and import the solution with the following commands in a command xml file:

```
<command xsi:type="command_python"
   cmdline="exportData('mymodel.dat')" />
<command xsi:type="command_solve">
  <solver name="lp" xsi:type="solver_lp">
     <loglevel>2</loglevel>
     <modelfile>mymodel.mod</modelfile>
     <datafile>mymodel.dat</datafile>
     <solutionfile>mymodel.sol</solutionfile>
     <minimum>true</minimum>
     <objective>column_name_1</objective>
     <objective>column_name_2</objective>
     <objective>column_name_3</objective>
  </solver>
</command>
<command xsi:type="command_python"
   cmdline="importSolution('mymodel.sol')" />
```

CHAPTER

7 Information for developers

This chapter discusses some topics of interest to developers working on extending, customizing or maintaining frePPLe.

1. Code structure
2. Class diagram
3. Extension modules
4. Portability
5. Version control
6. Style guide
7. Security
8. Internationalization

7.1 Code structure

This chapter provides a high level description of the code structure.
It provides brief notes that helps a developer find his/her way in the detailed C++ API reference and Class diagram .

Three layers can be distinguished:

- **Utility classes** which provide infrastructure-like services as a foundation for the next layers.
  - Object (p 89) as an abstract base class for all frePPLe objects.
  - Metadata (p 89) about objects.
  - Date, DateRange and TimePeriod (p 90) for dealing with dates and times.
  - Timer (p 90) for measuring execution time.
  - XML serialization (p 91) for reading and writing XML data.
  - Python binding (p 91) for interfacing with Python.
  - Command (p 91) for executing state changes.

  - Exception classes (p 90) for reporting error conditions.
  - Mutex (p 91) provides support for concurrent access to memory objects in a multihtreaded environment.
  - HasName and Tree (p 92) for representing entities with a name and storing them in a binary tree container.
  - HasHierarchy (p 92) allows objects be structured in a hierarchical tree, ie to refer to a parent and have children.
  - Leveled (p 92) for representing entities that are connected in a network graph.
- **Model classes** which represent the core modeling objects.
  See the chapter Modeling for the details.
  They are structured as a base class (or Category) with one or more concrete implementations (or Classes).
- **Extension classes** which inherit from the core model classes and implement specific new models or solver techniques.
  See the section Extension modules for more details.

7.1.1 Object

Object is an abstract base class.
It handles to following capabilities:

- Metadata: All subclasses publish metadata about their structure and the memory they consume.
- Concurrency: Locking of objects is required in multithreaded environments. The implementation of the locking mechanism is delegated to the LockManager class, and this class provides only a pointer to a lock object and convenience guard classes.
- Callbacks: When objects are created, changing or deleted, interested classes or objects can get a callback notification.
- Serialization: Objects need to be persisted and later restored.
  Subclasses that don't need to be persisted can skip the implementation of the writeElement method.

7.1.2 MetaData

FrePPLe uses a two level structure to group metadata:

- A **MetaCategory** represents an entity type. The metacategory will implement a container for all instances of this type, and also a handler method to control persistence of the objects.
  E.g. "Buffer"
- A **MetaClass** represents a concrete class. It belongs to a certain MetaCategory, and contains a factory method to generate objects.
  E.g. "BufferDefault", "BufferMinMax", "BufferInfinite"...
- **MetaData** is the abstract base class for the concrete class MetaClass and MetaCategory.

After creating an MetaClass or MetaData object it needs to be registered, typically in the initialization of the library.

7.1.3 Date - DateRange - TimePeriod

These classes allow easy and intuitive manipulation of dates, durations and date ranges.
The classes are implemented as a thin wrapper around the standard ansi C time functions and provides time accuracy of 1 second.

Durations are formatted according to ISO8601.

An example:

```
Date start = Date::now();
TimePeriod duration("P1D");
Date end = d + t;
DateRange dr(start, end);
cout << d << "  " << t << "  " << dr << endl;
```

The C library is respecting daylight saving time (DST). Depending on the timezone configured on your computer, you will have two days a year which last 23 or 25 hours instead of the regular 24 hours.
This means that "midnight on day 1" + "24 hours" will not always give you "midnight on day 2"!

7.1.4 Timer

This is a class to measure the excution time of the application with (at least) millisecond precision.
An example:

```
Timer t;
do_something();
cout << "something took " << t << " seconds" << endl;
t.restart();
do_something_else();
cout << "something else took " << t << " seconds" << endl;
```

7.1.5 Exception

FrePPLe uses 3 exception classes to report errors. Each of the classes inherits from std::exception.

- A **DataException** is thrown when data errors are found.
  The expected handling of this error is to catch the exception and allow the execution of the program to continue.
- A **RuntimeException** is thrown when the library runs into problems that are specific at runtime.
  These could either be memory problems, threading problems, file system problems, etc… Errors of this type can be caught by the client applications and the application can continue in most cases.
- A **LogicException** is thrown when the code runs into an unhandled and unexpected situation.
  The normal handling of this error is to exit the program, and report the problem. This exception always indicates a bug in the program code.

7.1.6 XML Serialization

The Object base class provides the following methods that need to be implemented by serializable clasess:

- The **beginElement** is called by the parser when reading the start of a tag.
- The **endElement** event is called by the parser when reading the end of a tag or attribute.
- The **writeElement** is called when serializing the object.

FrePPLe uses the SAX parser from Xerces-C to parse and validate input XML data.
The class **XMLImput** is a wrapper around the parser. It receives the SAX events and makes the appropriate calls to the frePPLe objects.
Subclasses are available to parse a file or a string.

Writing XML output is done with the **XMLOutput** class which provides methods to write a header, elements and attributes. Subclasses are available to write to a file or a string.

7.1.7 Python binding

A couple of utility classes are available to simplify the use of the Python C-api in the frePPLe C++ code.

- The **PythonObject** class handles two-way translation between the data types between C++ and Python.
- The template class **PythonExtension** is used to define Python extensions.
- The **PythonType** class is a wrapper around the type information in Python.
- The **PythonInterpreter** class maintains the Python interpreter.

7.1.8 Command

This class implements the design pattern with the same name. All state changes in the application are expected to be encapsulated in objects of this class.

The CommandList class works as a wrapper for a collection of other commands, following the classic composite design pattern.
This allows command hierarchies to be constructed, which can be executed in sequence or in parallel.

Quite a few subclasses are available: see the command modeling or the C++ API reference.

7.1.9 Mutex

Working with frePPLe in a multithreaded environment requires special control over concurrent acces to the objects in memory.

- **Mutex** allows exclusive access to a object.
  Depending on your platform it is implement as a thin wrapper around a Windows critical_section or as pthread pthread_mutex_t.
- **ScopeMutexLock** is a convenience class that makes it easy (and exception-safe) to lock a mutex in a scope.

- The **CommandList** (described above) has the capability to execute commands in parallel by spawning seperate threads.

7.1.10 HasName and Tree

The classes represent classes which use a std::string / name as a unique identifier.
The Tree class is implemented as a red-black binary tree, using HasName objects as the nodes (i.e. intrusive container).

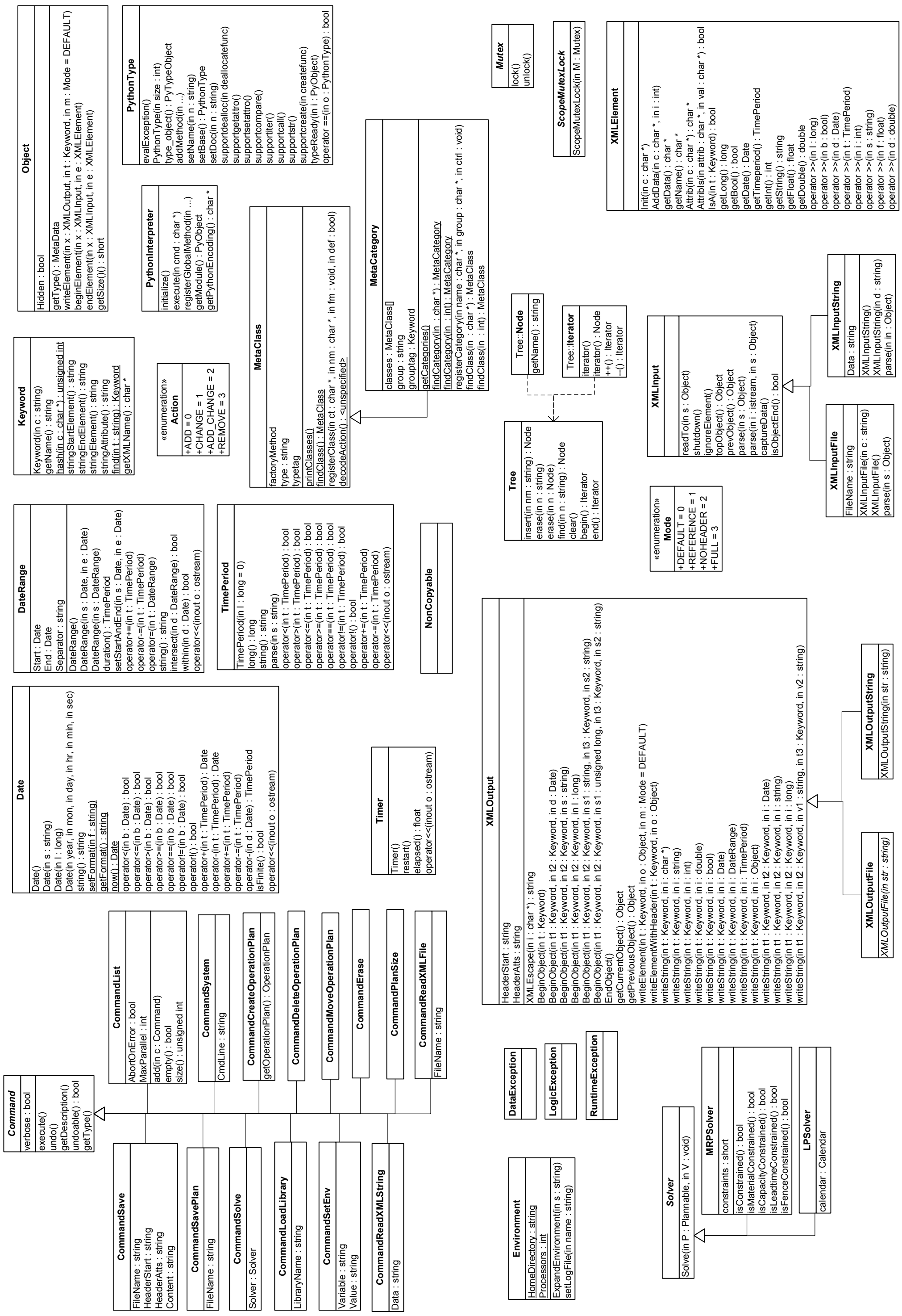
7.1.11 HasHierarchy

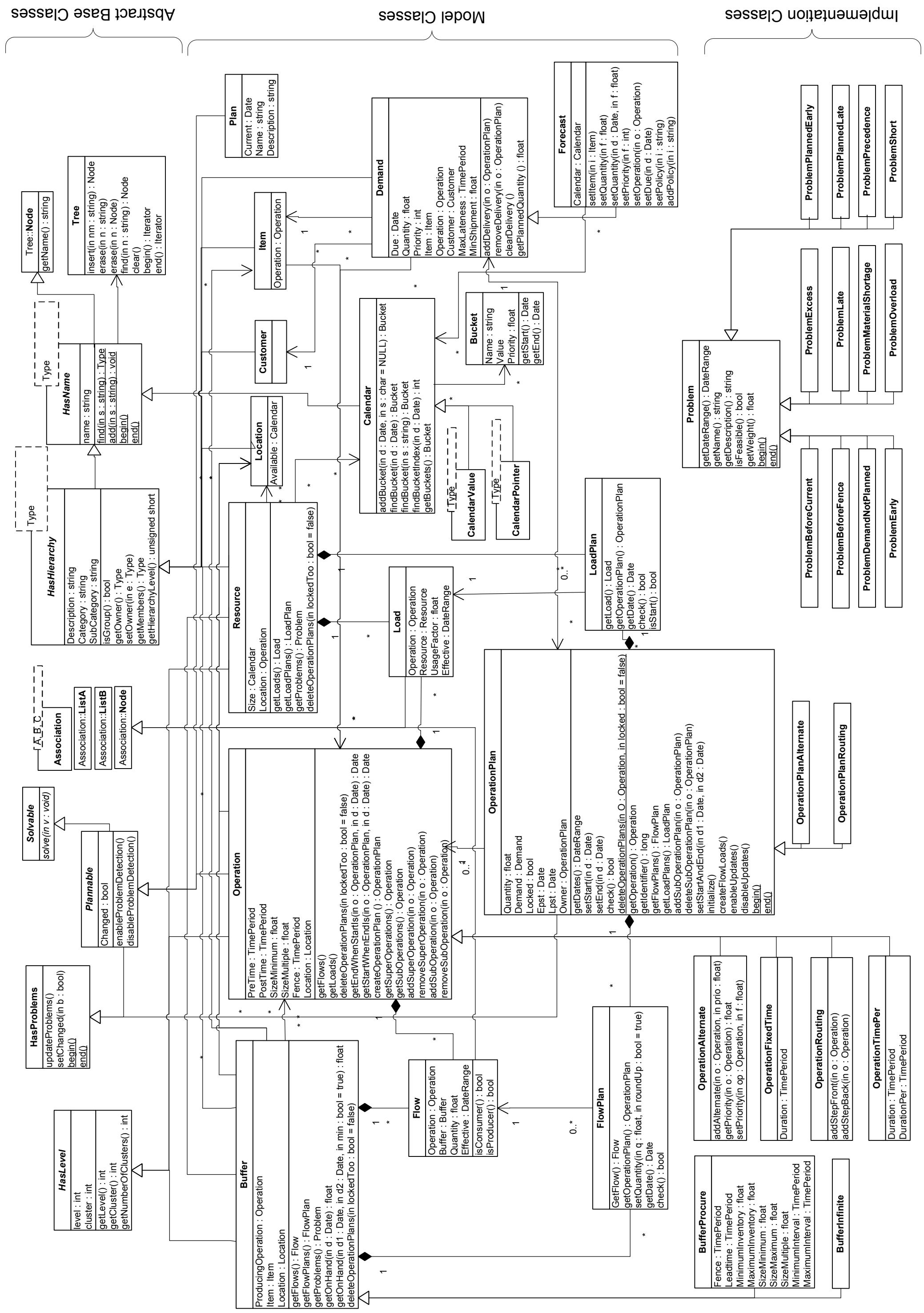
The class allows allows objects be structured in a hierarchical tree. A HasName object can point to a single parent and it maintains a linked list of children.

7.1.12 Leveled

The model classes Operation, Buffer, Resource, Load and Flow are the key objects that are used to represent the network.
The first three represent the actual entities, while Load and Flow represent associations/links between the entities.
See the section Cluster and level algorithm for the details.

7.2 Class diagram


Command
CommandList
CommandSystem
CommandCreateOperationPlan
CommandDeleteOperationPlan
CommandMoveOperationPlan
CommandErase
CommandPlanSize
CommandReadXMLFile
CommandSave
CommandSavePlan
CommandSolve
CommandLoadLibrary
CommandSetEnv
CommandReadXMLString
Date
Timer
DateRange
TimePeriod
NonCopyable
Keyword
Action
MetaClass
Object
PythonInterpreter
PythonType
MetaCategory
Environment
DataException
LogicException
RuntimeException
XMLOutput
XMLOutputFile
XMLOutputString
Solver
MRPSolver
LPSolver
Tree
Mode
Tree::Node
Tree::Iterator
XMLInput
XMLInputFile
XMLInputString
Mutex
ScopeMutexLock
XMLElement


Abstract Base Classes
Model Classes
Implementation Classes
Tree::Node
Tree
HasName
HasHierarchy
Association
Solvable
Plannable
HasProblems
HasLevel
Plan
Demand
Forecast
Item
Customer
Location
Calendar
Bucket
CalendarValue
CalendarPointer
Resource
Load
LoadPlan
Operation
OperationPlan
Buffer
Flow
FlowPlan
Problem
ProblemPlannedEarly
ProblemPlannedLate
ProblemPrecedence
ProblemShort
ProblemExcess
ProblemLate
ProblemMaterialShortage
ProblemOverload
ProblemBeforeCurrent
ProblemBeforeFence
ProblemDemandNotPlanned
ProblemEarly
OperationPlanAlternate
OperationPlanRouting
OperationAlternate
OperationFixedTime
OperationRouting
OperationTimePer
BufferProcure
BufferInfinite

7.3 Extension modules

FrePPLe is designed as an extendable framework.
Additional modeling and solver modules can be loaded at runtime without recompiling the library. Such extension modules can be shipped with frePPLe, or can be developed by third parties. Modules can be open source or have a commercial license.

An simple example is available in the testcase sample_module.
FrePPLe currently includes three examples of such extension modules: a module implementing a python interpreter, a forecast class implementing a special type of demand, and a solver using a linear programming algorithm.

The steps below define how a custom extension can be build on the framework.

- The proper way to build extension is by creating modules.
  Other ways of extending the package may technically be possible, but are not recommended. Copying the code and header structure from an existing module is the quickest and easiest start.
- Create your own header files, and include the frePPLe header file planner.h to have access to the frePPLe objects.
  A simple header file can look like this:

```
#include "frepple.h"
using namespace frepple;

namespace your_module
{
   MODULE_EXPORT const char* initialize(
     const CommandLoadLibrary::ParameterList& z
     );
   ...
   your classes and function definitons
   ...
}
```

- Create your own c++ implementation files, which will include you customized header file.
  It is important is to include an initialize() method, and use it to register your extension in the frePPLe framework. The method is automatically called when the module is loaded.

```
#include "your_module.h"
namespace your_module
{

MODULE_EXPORT const char* initialize(
  const CommandLoadLibrary::ParameterList& z
  )
{
  ...
  your initialization code goes here
  ...
}
```

```
your method and class implementations go here
```

- Compile your code as a loadable module.
  The command line options and arguments vary for each compiler and platform. For gcc I use the options "-module -shrext .so -avoid-version -rpath /dev/null", adding also "-no-undefined" when running under cygwin.
  To keep things simple and transparant please use the .so extension for you modules and place them in the $FREPPLE_HOME directory.
- Update the $FREPPLE_HOME/init.xml file to load your module with a COMMAND_LOADLIBRARY tag.
  Parameters specified with the PARAMETER tag are passed to the initialize() funtion when the module is loaded.
- Update the file $FREPPLE_HOME/frepple.xsd by defining the xml constructs enabled by your module.
  To keep things clean and modular, it is recommended to do this by including a seperate xsd file rather than directly entering the definition in the file.

7.4 Portability

The project is compiled and tested only for 32-bit and 64-bit linux and Windows environments, with Linux being the primary development platform. Porting to other platforms is encouraged - you'll have all my support in helping with this.

Here are some areas where porting may be a bit challenging:

- Availability of a modern C++ compiler and STL.
- File system functions such as fstat, paths, directory listings
- Availability of the Pthreads library for threading.
  FrePPLe currently only supports the Windows threading functions and the Pthreads.
- Shared libraries
  Currently the code only supports the dlopen (Solaris, Linux and various BSD flavors) and LoadLibrary (Windows) functions. HPUX uses different function name shl_load, while AIX seems to use a more exotic mechanism.
- Availability of the Xerces-C XML parser.
- Availability of the Python language.

7.5 Version control

The software changes are tracked with subversion on the Sourceforge site.
The subversion repository allows anonymous access. Use the following command to checkout the latest version of the code:

svn co https://frepple.svn.sourceforge.net/svnroot/frepple/trunk frepple

The repository content can also be browsed online at http://frepple.svn.sourceforge.net/viewvc/frepple/

Complete detailed instructions are available on http://sourceforge.net/svn/?group_id=166214.

A example subversion configuration is available in the file subversion.config for convenience. In particular the section on the automatic properties is of interest when adding files to the project.

7.6 Style guide

To enforce the same formatting of the source code the astyle tool is used.
See http://astyle.sourceforge.net/ for more information.

The following formatting options are used:

```
-- style=ansi
-- indent=spaces=2
-- indent-classes
-- indent-switches
-- min-conditional-indent=2
-- one-line=keep-statements
-- one-line=keep-blocks
-- max-instatement-indent=2
-- convert-tabs
```

Astyle does a pretty decent job, but reviewing the astyle changes before committing them is still required: astyle sometimes misses the point…

7.7 Security

When frePPLe is used in a networked multi-user environment, security is very important.
The frePPLe C++ code is developed with security in mind.

Here are some notes and considerations on this topic:

- FrePPLe can validate incoming XML data with an XML-schema. Invalid data will be rejected and an error message is generated.
  The default xsd files frepple.xsd and frepple_core.xsd cover all valid structures.
  When integrating frePPLe with other systems it is strongly recommended to validate the incoming XML data against a small and well-controlled subset of the default XML-schema.
- The COMMAND_SYSTEM command allow execution of arbitary shell commands with the privilege of the user running the Frepple executable.
  While allowing a maximum of flexiblity for configuring and customizing Frepple, it also creates an open door to access your system. Access to this command should be restricted, and/or frePPLe should be run by a user account with limited privileges.
- The COMMAND_PYTHON command and the PYTHON XML processing instruction allow execution of arbitrary python statements with the privilege of the user running the frePPLe executable.
  While allowing a maximum of flexiblity for configuring and customizing frePPLe, it also creates an open door to access your system. Access to this command should be restricted, and/or frePPLe should be run by a user account with limited privileges.
- The COMMAND_SETENV command allows environment variables to be updated.
  Access to this command should be restricted, as it can alter the behavior of the system.

- When using Django, its standard web authentication mechanism is relatively weak.
  In secure environments, consider using HTTPS and plugging in a different authentication mechanism.

7.8 Internationalization

This section contains some notes on topics relevant for the internationalization.

1. **It is highly recommended to use utf-8 as the encoding of character data**.
   Using it consistently for your **locale**, **XML-files** and **databases** helps in avoiding headaches and sleepless nights.
2. When creating a database for the Django user interface, make sure the character encoding properly support utf-8.
   When using **MySQL**, this is easiest don by setting the database parameter "default character set" to "utf-8" and "default collate" to "utf8_general_ci".
   When using **Oracle**, this is controlled through the database "character set" and "national character set".
   PostgreSQL provides the 'encoding' setting on the database.
   SQLite is unicode-ready by default.
3. Xerces-C will transcode the input XML data from the input encoding (typically specified with a `<?xml version="1.0" encoding="UTF-8" ?>` header line) to the locale of your *nix shell or Windows environment.
   Xerces-C has intrinsic support for ASCII, UTF-8, UTF-16 (Big/Small Endian), UTF-32(Big/Small Endian), EBCDIC code pages IBM037, IBM1047 and IBM1140 encodings, ISO-8859–1 (aka Latin1) and Windows-1252.
   This means that it can parse input XML files in these above mentioned encodings. For more exotic encodings a special configuration and compilation is required: see the Xerces-C documentation for more details.
4. Internally frePPLe stores string data in the **locale** of your environment: see the documentation on the setlocale C function.
   For most modern Linux distributions the default setting is a utf-8 encoded locale, meaning that every unicode character can be represented. The environment variable LC_ALL can be used to specify a suitable locale.
   On windows the default locale is some ANSI default codepage.
5. When exporting data out of frePPLe, no data conversion to specific encodings is done.
   All output will be in the locale of your environment.
6. FrePPLe internally uses byte-based string manipulation routines, not character-based.
   For utf-8 encoding and the single-byte codepages this works fine, but with multi-byte encodings such **utf-16** and **utf-32** this won't work any more. Such encodings are NOT supported by frePPLe.

CHAPTER 8

Unit tests

These pages document the test suite available in the 'test' subdirectoy. The tests can be categorized in the following functional categories:

- Unit tests, which verify the behavior specific parts of the code.
- Performance tests, which focus on the performance (memory and/or cpu-time).
- Samples, which demonstrate the real-life usage of the tool.

The test suite is run by the **runtest.py** script in the test subdirectory. You need to have Python installed on your machine to run the test suite.
Example usage:

runtest.py:
```
Run all tests
```
runtest.py -vcc:
```
Run all tests on Windows
```
runtest.py A B:
```
Run the tests A and B
```
runtest.py -- debug A:
```
Run the test A, verbosely showing its output
```
runtest.py -- help:
```
Print information on the script and its options
```

The tests described here only test the core library.
A seperate test suite exists for the Django user interface.

1. Test Callback
2. Test Cluster
3. Test Command 1
4. Test Command 2
5. Test Constraints Leadtime 1
6. Test Constraints Material 1

7. Test Constraints Material 2
8. Test Constraints Material 3
9. Test Constraints Resource 1
10. Test Constraints Resource 2
11. Test Constraints Resource 3
12. Test Constraints Resource 4
13. Test Constraints Resource 5
14. Test Datetime
15. Test Deletion
16. Test Demand Policy
17. Test Flow Effective
18. Test Forecast 1
19. Test Forecast 2
20. Test Forecast 3
21. Test Forecast 4
22. Test Forecast 5
23. Test Load Effective
24. Test LP Solver 1
25. Test Name
26. Test Operation Effective
27. Test Operation Pre Op
28. Test Operation Routing
29. Test Pegging
30. Test Python 1
31. Test Python 2
32. Test Python 3
33. Test Problems
34. Test Procure 1
35. Test Safety Stock
36. Test Sample Module
37. Test Scalability 1
38. Test Scalability 2
39. Test Scalability 3
40. Test XML
41. Test XML Remote

8.1 Test Callback

This test verifies the event publishing and subscription mechanism.

8.2 Test Cluster

This test verifies the correctness of the clustering algorithm.
A network is built with a whole range of possible interconnections between operations, buffers and resources.

8.3 Test Command 1

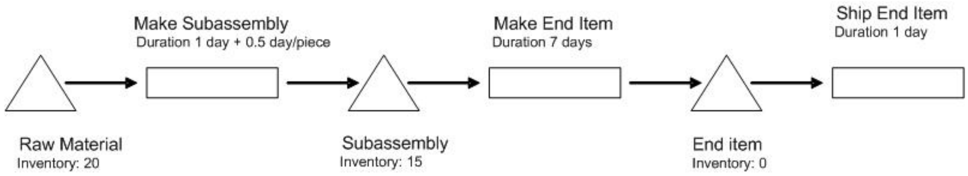
In this test commands are being run in parallel and in sequence. The proper branching and merging of the tasks is verified, and the behavior in case of errors.

8.4 Test Command 2

Verifies how environment variables are set and their values expanded.

8.5 Test Constraints Leadtime 1

This test verifies the solver behavior for leadtime constraints. Demands are placed on the network such that operations are planned in the past in the unconstrained plan. Demands are appropriately shorted or planned late in the constrained plan to solve the problems.



A first order for 7 units is due on day 3 after the current date.
It is planned to be delivered late on day 8: the production of the end item starts on the current date, and takes 7 days. The delivery takes an additional day.

A second order for 14 units is due on day 11.
The inventory of the subassembly is now depleted and 6 new subassemblies need to be produced. These subassemblies are due on day 3.
In the 2 days between the current date and the due date of the subassemblies 2 units can be produced. There are 3 subassembly operations are planned in parallel, each for 2 units, starting on the current day and finishing on day 3.
Sufficient raw material is available in inventory for the subassemblies.
The order is delivered on time.

8.6 Test Constraints Material 1

This test verifies the behavior of the buffer solver for the case where no producing operation is defined.
Four variations of a base scenario are tested:

- 3 consumers, ordered in chronological order
- 3 consumers, not ordered in chronological order
- extra supply arriving at a different date, causing a late order
- extra supply arriving at a different date, but already partially used up

8.7 Test Constraints Material 2

@todo

8.8 Test Constraints Material 3

@todo

8.9 Test Constraints Resource 1

A simple capacity problem that can be resolved by moving operation plans early.

8.10 Test Constraints Resource 2

A capacity shortage where operation plans are moved earlier till they are in the past.
The associated demands are then shorted.

8.11 Test Constraints Resource 3

A capacity problem where a single operation loads multiple resources. This test case also has capacity limits varying over time.

8.12 Test Constraints Resource 4

This test shows how capacity constraints are solved in situations with a complex load profile and with interaction between material and capacity constraints.
A few solver loops are required to fill the available capacity slots and minimize the lateness.

8.13 Test Constraints Resource 5

In this test the resource capacity varies heavily over time.
The test case verifies the resource solver is capable of using every single bit of capacity available on the resource. The capacity search is done for two situations: once with a search backward in time, and another one forward in time.

This test also verifies the logic used by calendars to select the bucket that is in effect on a certain day.

8.14 Test Datetime

FrePPLe uses some wrapper classes around the C date and time functions.
These are tested here: conversions to and from strings, additions, ...

8.15 Test Deletion

This test verifies the capability to delete parts of the model. After loading the model different entities are one-by-one being deleted. After each delete we replan and save the model to make sure the deletion is working correctly: an incorrect delete would crash the application!

8.16 Test Demand Policy

The test verifies the demand policies.

The supply situation is such that half of the demand can be met in time, and half of it late:

- Demand: 20 on due date 5 Jan
- Supply: 10 available as inventory, and 10 arriving on 10 Jan

The demand policy controls how the demand is allowed to be planned in such a constrained situation:

- Case A:
  The default policy is to allow demands to be planned without any limits on the timing and quantity of the deliveries.
  Result: Delivery of 10 units on 5 Jan and a second delivery on 10 Jan.
- Case B:
  No lateness is allowed.
  Result: A delivery of 10 units on 5 Jan.
- Case C:
  Lateness is allowed, but we only accept a delivery for the full requested quantity.
  Result: A delivery of 20 units on 10 Jan.
- Case D:
  No lateness is allowed, and we also only accept a delivery for the full requested quantity.
  Result: No delivery planned.
- Case E:
  The maximum allowed delivery date is jan 7, without any restriction on the delivered quantity.
  Result: A delivery of 10 units on 5 Jan
- Case F:
  The minimum quantity shipped is 11, without any restriction on the delivery date.
  In this case the onhand on jan 5 is increased to 15.
  Result: A delivery of 20 units on 10 Jan

8.17 Test Flow Effective

This test verifies the behavior of date effective flows:

- case 1: effectivity on consuming flows of a delivery operation
  This scenario models a situation where an old product is being replaced by a new version starting from a certain date.
- case 2: date-effective material consumption with constrained supply
  The supply of date effective component A is constrained. Extra supply arrives only after the end of the effectivity of the component. This extra supply is ignored since the flow is not effective any more at that time.
- case 3: date-effective producing flow
  This scenario models a so-called learning curve: the production of a new product becomes more efficient as time progresses.
  The operation "3. make end item" produces a variable number of units of the end item. In january it produces 0.7 units, in februari it produces 0.8 units and from then onwards it produces 1.0 units.

8.18 Test Forecast 1

The first step in the forecast netting process is associating each actual order with a forecast it can net from.
This test case test this matching algorithm.

A customer hierarchy is modeled as follows: "grandparent customer" > "parent customer" > "customer".
An item hierarchy is modeled as follows: "grandparent item" > "parent item" > "item".
Forecasts are defined at various combinations of these levels.
Actual orders are then looking for a matching forecast in these hierarchies.

Different scenarios are being validated:

- A: an order matches a forecast at 'customer'+'item' level
- B: an order matches a forecast at 'item' level
- C: an order matches a forecast at 'parent customer' + 'item' level
- D: an order matches a forecast at 'customer' + 'parent item' level
- E: an order matches a forecast at 'parent customer' + 'parent item' level

8.19 Test Forecast 2

This test verifies the forecast distribution functionality.
This functionality allows specifying the forecast for a certain date range. FrePPLe then breaks it down into smaller time buckets that are used for planning.

This functionality is typically used to translate between the time granularity of the sales department (which creates a sales forecast per e.g. calendar month) and the manufacturing department (which creates manufacturing and procurement plans in weekly or daily buckets).
Another usage is to model a delivery date profile of the customers. Each bucket has a weight that is used to model situations where the demand is not evenly spread across buckets: e.g. when more orders are expected due on a monday than on a friday, or when a peak of orders is expected for delivery near the end of a month.

Two example scenarios are tested:

- The forecast value is specfied for a date range of 4 weeks.
  For planning in frePPLe the forecast is automatically spread over 21 daily buckets and a weekly bucket. Among the daily buckets, saturdays and sundays don't get any forecast. Also, mondays are busier than fridays and get a bigger share of the forecast.
- The forecast value is specified in calendar months. For planning in frePPLe the forecast is spread over weeks.
  Since the week and month boundaries don't align, the forecast is proportionally split across all intersecting weeks.

8.20 Test Forecast 3

This test verifies the forecast netting behavior.
Actual orders are searching a matching forecast, and then look for available net forecast in the forecast buckets. The search for net forecast first looks backwards in time and then forward in time, respecting the parameters Net_Early and Net_Late which define the allowed time fence.

The test also verifies that the saved xml-file can be read in again at a later stage, producing an identical model.

8.21 Test Forecast 4

This test verifies how the forecast distribution works with discrete forecasts.

A forecast of 1 over a date range of 28 daily buckets will result in a 0 zero forecast for all days, except for the middle one.
A forecast of 2 over the same date range will give 2 buckets with a forecast of 1: a first one on the 7th day and a second one on the 21st day.
The test case has a couple more examples on the above.

8.22 Test Forecast 5

A number of cases are tested for the forecast generation based on a time series of historical data.

- A simple constant demand
- A simple trended demand
- A very irregular demand
- The historical demand is first trended and then constant
- The historical demand is first constant and then a trend starts
- A forecast with a seasonal demand pattern
- A forecast with very little historical data

8.23 Test Load Effective

This model verifies the behavior of date effective loads:

- case 1: unconstrained situation where operationplans intersect in various ways with the effective period.
- case 2: similar to 1 but with a capacity constraint, which is solved by producing early.
- case 3: similar to 1 but with a capacity constraint, which causes demand to be satisfied late.

8.24 Test LP Solver 1

This test shows how the linear programming solver is used to solve a capacity allocation problem in an optimal way.

The problem input consists of:

- A set of time buckets.
- A set of demands, each with a due bucket, a quantity and a priority.
- A set of resources, each with an available capacity per time bucket.
- A set of loads, i.e. demands requiring some time on one or more resources.

The problem is subject to the following constraints:

- For each time bucket and each resource:
  sum of capacity used by each demand <= capacity available in the resource bucket
- For each demand:
  sum of planned quantities in different buckets <= requested demand quantity

The LP problem solves for a hierarchy of goals.

- Minimize the shortness of demand of priorities 1, 2 and 3
- Minimize the lateness of demand of priorities 1, 2 and 3
- Minimize the early use of capacity (ie use capacity before the due date)

8.25 Test Name

This test reviews the data structure that is used for storing all named entities: functionality of the insertion, deletion and search operations, as well as their scalability The time for these operations properly fits a logaritmic profile, as expected with a binary tree data structure. A testing routine for this profile is also included in the test, but it isn't part of the regression tests since it isn't easy to produce a good pass-fail criterion.

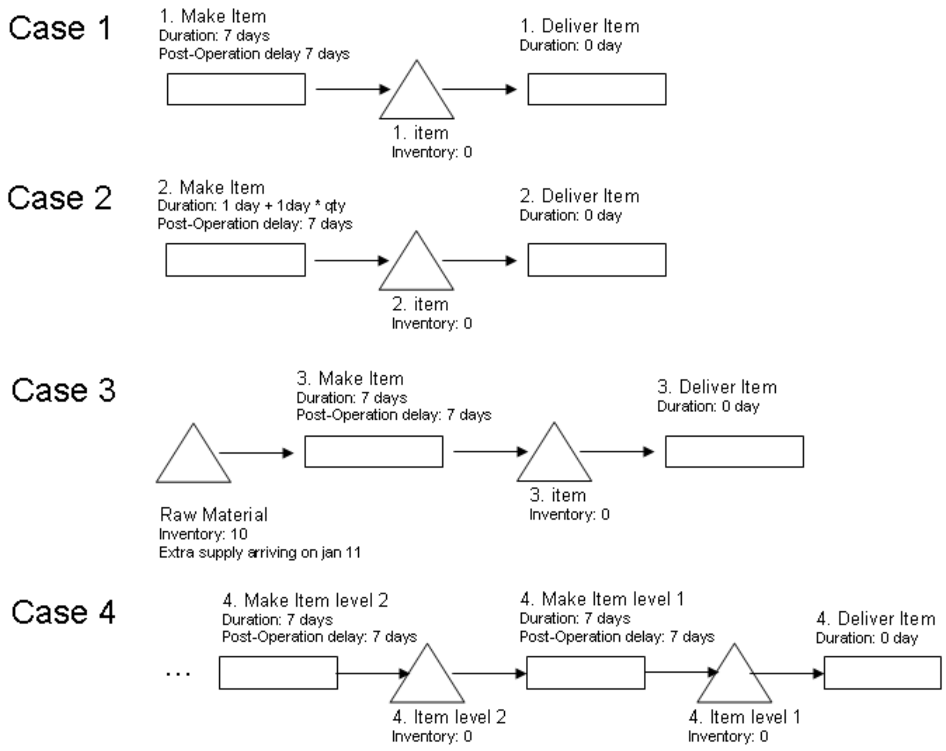
8.26 Test Operation Effective

This test checks the code for selecting of date effective alternate operations.

The test resolves about a product that can be sourced from two locations. In a first part of the horizon location A is the only allowed source, while in the last part of the horizon only sourcing from location B is possible. A transition period exists where sourcing from both location is possible.

8.27 Test Operation Pre Op

This test verifies the behavior of pre-operation and post-operation delays.
These are as delay times before and after an operation, which the solver tries to respect but can violate if required.



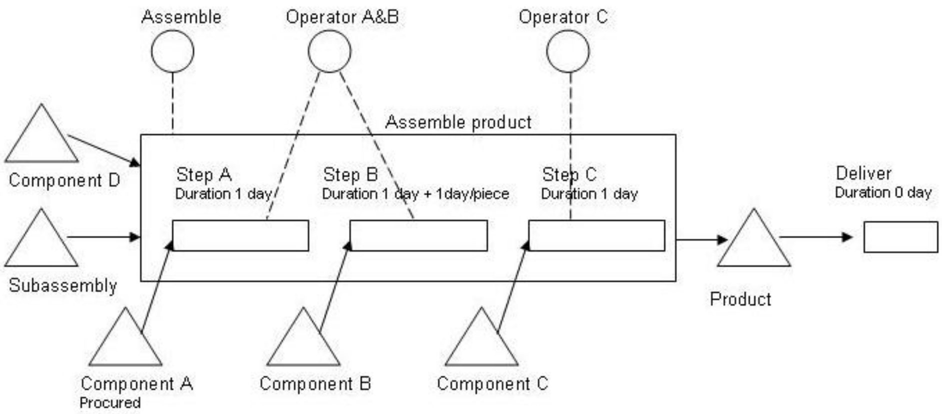
Several cases are included in this test:

1. Post-operation time on a fixed-time operation.
   The post-operation time is respected when possible, but when running against a leadtime constraint the post-operation time is reduced to meet the demand on-time / asap.
2. Post-operation time on a time-per operation.
   The constraint is again a leadtime constraint.
3. Post-operation time on fixed-time operation.
   This time the constraint is the late supply of raw material supply. It causes the post-operation time to be reduced.
4. Post-operation time on multiple levels in the supply path.
   The supply path is four levels deep, and a post-operation time is set at each level.
   In case of material of leadtime constraints the post-operation time on the most upstream operation/operations (i.e. operations deeper in the bill of material) is/are shrunk first.

8.28 Test Operation Routing

A routing operation is is built up from a number of suboperations that are executed in sequence. This test verifies the behavior of routing operations.

The test plans the routing with different material, capacity and leadtime constraints.



8.29 Test Pegging

Verifies the correctness of the material pegging. Material streams are traced upstream and downstream and printed to the output.

8.30 Test Python 1

This test verifies and demonstrates the embedded Python interpreter.
It verifies:

- Executing Python code as XML processing instruction.
- Executing Python code in a seperate source file.
- Performance comparison of data loading in different ways.
- Catching of exceptions thrown from frePPLe C++ code.
- Executing Python code in different threads

No pass/fail criterion is present in this test.

8.31 Test Python 2

This test shows how we can access frePPLe objects from Python.

8.32 Test Python 3

This test shows how we can use Python to create a frePPLe model: we can create objects, access existing objects and change objects.

8.33 Test Problems

Verifies that problems objects are created and deleted properly when the model is being updated in various ways.

8.34 Test Procure 1

This unit test verifies the behavior of procurement buffers in a number of scenario's.
The different cases are:

1. Base scenario.
2. Procure in multiples.
3. Procurement with miniumum size, maximum size and in multiples.
4. Invalid parameters for size constraints.
5. Procurement with minimum and maximum interval.
6. The full monty. Procurement with minimum interval, maximum interval, miniumum size, maximum size and in multiples.
7. Procurement with fixed interval.
8. Procurement in fixed quantity.
9. Procurement in fixed quantity with fixed interval.

In all these cases the demand is directly placed on the procured item (i.e. no bill of of material is involved at all) and the demand pattern is also identical.
The test runs first an unconstrained plan, followed by a constrained plan.

8.35 Test Safety Stock

This test demonstrates the capabilities to model and plan safety stocks in frePPLe.

There are 2 ways:

1. Quantity-based safety stock.
   A minimum calendar on a buffer defines the desired minimum stock level, which can vary over time.
   The solver tries to replenish to this level when replenishing the buffer, but handles it as a soft constraint only.
   The buffer flags a problem when the inventory drops below the minimum target.
2. Time-based safety stock.
   A post-operation time on an operation defines a time delay after the end of the operation.
   The solver tries to respect this delay, but handles it as a soft constraint only.
   No problem is shown when the post-operation time is shrunk or reduced.

8.36 Test Sample Module

A simple example on how to define an extension module for Frepple.
The example defines a new operation type that can be used to represent transportation operations easier.

8.37 Test Scalability 1

Tests the scalability of the data loading, running an MRP plan (including the clustering algorithm) and saving the plan. The network in this case consists of a lot of parallel clusters, which can be solved in parallel. See also the test scalability_2

The algorithms scale linearly with the model size, while the mayor underlying data structures are binary trees which scale logarithmically with the model size... The result is a runtime that combines both. In summary, one could say that the system scales a bit worse than linear, but definately not quadratic or worse.

8.38 Test Scalability 2

In this test a model is created based on parametrizable values of:

- Number of clusters.
- Number of demands per cluster.
- Depth of the supply chain, i.e. number of levels.

Comparing the runtime with different values of these parameters allows to gain a better understanding of the factors that are impacting memory and runtime most significantly

The algorithms scale linearly with the model size, while the mayor underlying data structures are binary trees which scale logarithmically with the model size... The result is a runtime that combines both. It depends on the data set, the platform and the compiler how your model will scale.

8.39 Test Scalability 3

This test is designed to verify the scalability of the timeline data structure. The network consists of a single buffer with a very simple operation producing into it. Since the timeline data structure is currently based on a linear list the scalability of the timeline is expected to be bad... A quadratic increase in the runtimes can be observed A more scalable data structure has been designed to provide a more scalable implementation.

8.40 Test XML

This is a test for the XML parser routines. The test consists of a complex xml document to be parsed and processed:

- XML tags 8 nested levels deep
- ignore-element sections

8.41 Test XML Remote

This test uses the HTTP protocol to pick up XML-data from the URL http://frepple.sourceforge.net/test/xml_remote.xml.
The test is implemented using the urllib2 Python library.

CHAPTER

9 Appendices

1. GNU Lesser General Public License
2. GNU Free Documentation License

9.1 GNU Lesser General Public License

Version 2.1, February 1999

Copyright © 1991, 1999 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111–1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages—typically libraries—of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

> a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
>
> b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agree-

ment or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE

ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

9.2 GNU Free Documentation License

Version 1.2, November 2002

Copyright © 2000,2001,2002 Free Software Foundation, Inc. 51 Franklin St, Fifth Floor, Boston, MA 02110–1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright ©  YEAR  YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with... Texts." line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.