

Energy Technologies

Documentation



Version 2.1, Revision 2455

December 18, 2012

Leni systems development team



Laboratoire d'Energétique Industrielle (LENI)

Laboratoire d'Energétique Industrielle
ME A2 434 (Bâtiment ME)
Station 9
CH-1015 Lausanne
Tél.: +41 21 693 35 06
Fax: +41 21 693 73 22
 <http://leni.epfl.ch>
 secretariat.leni@epfl.ch

This guide and the EnergyTechnologies syntax has been created by:

Raffaele Bolliger raffaele.bolliger@epfl.ch

The following people have contributed to EnergyTechnologies development and documentation:

Helen Becker
Zoé Périn-Levasseur
Matthias Dubuis
Martin Gassner
Hubert Thieriot
François Maréchal
Dave Bergomi
Léda Gerber
Samira Fazlollahi
Nicolas Borboën
Cédric Blondel

... and probably most of the students and researchers at LENI.

Contents

1	What's new	8
1.1	Version 1.2	8
1.1.1	EI entities	8
	Layers	8
	Streams	8
	Locations	8
2	Introducing EnergyTechnologies	9
2.1	Introduction	9
2.2	What is EnergyTechnologies	9
2.3	Programming language	10
2.4	What is a model	10
2.5	Extension of the generic model - Methods interfaces	11
2.6	Programming strategy	11
2.7	Running models	12
2.8	Models database - EnergyTechnologies Database	12
3	EnergyTechnologies syntax	13
3.1	Variable types for fields	13
3.2	Elements identifier - TagName	14
3.3	Streams and Internal Parameters	14
	3.3.1 Material and Energy Streams	15
	3.3.2 Internal parameters	15
3.4	System synthesis and analysis methods interfaces	16
3.5	Model files and links to other software	18
3.6	Version and compatibility	18
3.7	Model Classification	19
	3.7.1 Grouping	19
	Classification tree	20
	3.7.2 Keywords	21
	3.7.3 Stream Connections	21
	3.7.4 Modeling detail	22
3.8	Authoring and Bibliography	22
	3.8.1 Authoring	23
	3.8.2 Bibliographic references	23
3.9	Documentation	24
3.10	Validation	25
4	Method Interfaces Syntax	27
4.1	Why Method Interfaces?	27
4.2	How do they work?	27
4.3	Officially supported method interfaces	28

5	Model Validation	29
5.1	Validation status	29
5.2	Rating procedure	30
5.3	Model Description	30
5.4	Bibliographical sources	30
5.5	Model robustness and validity range	30
5.6	Model calibration on measurements and/or other models	30
5.7	Peer review	30
6	Model Usage	31
6.1	<i>OSMOSE</i> - Running the models	31
6.2	Run a model in its default state	31
6.3	Modify model parameters	32
6.4	Modify Method interfaces parameters	33
7	EnergyTechnologies Database	35
7.1	Implemented technologies	35
7.2	Database Usage	36
7.2.1	Technologies Identification	36
7.2.2	Technology search with classification method - <code>identifyMFielName</code>	36
7.2.3	Information about the database - <code>EnergyTechnologiesInfo</code>	36
7.2.4	Browse the database - <code>browse_db</code>	37
8	Input and Output Streams Database	38
8.1	Basic syntax	38
8.2	Extended syntax	38
8.3	Streams types	38
8.3.1	Fuels	38
	Fuel types	39
	Fuel State	39
	Chemical Properties	39
	Composition	40
	Matlab code for Fuels	40
8.4	Streams database	40
A	Process Integration	41
A.1	What is energy integration?	41
A.2	System methods interface syntax	41
A.2.1	Hierarchy - Containers and content	42
A.2.2	Generic fields for all the syntax elements	42
A.3	Layers	43
A.4	Units	44
A.4.1	Generic fields for energy conversion units	45
A.4.2	Process units and utilities - <code>process SYNPRO</code> , <code>utility SYNCCY2</code> and <code>SYNUT2</code>	47
A.4.3	Rankine cycles - <code>SYNRNK</code>	47
A.4.4	Refrigeration and heat pump cycles - <code>SYNFRG</code>	48
A.4.5	Steam network units - <code>SYNHEAx</code>	51
A.4.6	Fuels - <code>SYNFUEL</code>	54
A.4.7	Steam network parameters - <code>SYNUNT</code>	54
A.4.8	Easy parameters - <code>EASYPRFS</code>	54
A.4.9	Costing unit - <code>SYNCOST</code>	55
A.4.10	Investment cost parameters - <code>SYNINV</code>	55
A.4.11	Operating cost parameters - <code>SYNOPC</code>	57
A.5	Groups	58
A.5.1	Steam Networks - <code>SteamNetwork</code>	59

A.6	Streams	60
A.6.1	Overview of available streams	60
A.6.2	Note about layers	61
A.6.3	Thermal Streams - <code>qt</code>	61
A.6.4	Variable Thermal Streams - <code>vqt</code>	62
A.6.5	Piecewise thermal Streams - <code>ht</code>	62
A.6.6	Fuels - <code>fuel</code>	63
A.6.7	Steam - <code>steam</code>	64
A.6.8	Variable Steam Streams - <code>vsteam</code>	65
A.6.9	Common substances streams - <code>stream</code>	66
A.6.10	Cost Streams - <code>cost</code>	68
A.6.11	Mass Streams - <code>mass</code>	68
A.6.12	Mass Streams with quality - <code>mq</code>	69
A.6.13	Mass Streams with partial flowrates - <code>mqpartial</code>	69
A.7	EI Variables	71
A.8	Constraints and Equations	72
A.8.1	Adding terms to the equations - <code>EqTerms</code>	73
A.9	Heat exchange connection constraints	74
A.9.1	Heat exchange restrictions between units	74
A.9.2	Heat exchange restrictions between sub-systems	75
A.10	Locations	76
A.11	Networks	76
A.12	Storage: SYNSTO	76
A.13	Multi time, Multi periods	78
B	Emissions	80
C	Costs	81
	Reference year <code>Reference_year</code> , [-]	81
	Marshall&Swift Index, <code>MS_Index</code>	81
	Bare Module Cost, <code>BM</code> , [€ \$ CHF]	81
	Total Module Cost, <code>TM</code> , [€ \$ CHF]	81
	Specific Grass Root, <code>SpecificGR</code> , [€ \$ CHF]	81
	Unit, <code>Unit</code> , [€ \$ CHF]	81
	Maintenance, <code>Maintenance</code> , [€ \$ CHF]	81
	Operation, <code>Operation</code> , [€ \$ CHF]	81
	α_1 , <code>f_TotalModule</code> , [-]	81
	α_2 , <code>f_GrassRoot</code> , [-]	81
D	Life Cycle Assessment	83
D.1	What is Life Cycle Assessment?	83
D.1.1	Definitions	85
D.2	Specificities of LCA for components of process equipment	87
D.3	Multi-period and Life Cycle Stage modes	88
D.3.1	Definition of the life cycle stages	88
D.4	Method interface syntax	89
D.4.1	Generic fields for all the syntax elements of LCI	89
D.5	Unit Processes	89
D.5.1	Generic fields for unit processes	90
D.5.2	Integrating new datasets	90
D.6	Elementary Flows	91
D.6.1	Generic fields for elementary flows	91
D.7	Components	91
D.7.1	Generic fields for components	91
D.7.2	Boiler	93

D.7.3	Compressor	93
D.7.4	Engine	93
D.7.5	Filter	94
D.7.6	FlashDrum	94
D.7.7	HeatEx_ST	95
D.7.8	Molecular_Sieve	95
D.7.9	Pump	95
D.7.10	Reactor_SS	96
D.7.11	Tank_Concrete	96
D.7.12	Turbine	97
D.8	User-defined parameters	97
	Functional Unit definition	98
	Graphics parameters definition	98
	General Life Cycle Parameters	99
D.8.1	Impact assessment methods	100
	Cumulative Energy Demand	101
	CML 2001	101
	Ecoindicator99	103
	Ecological Footprint	103
	Impact 2002+	103
	IPCC - climate change	104
	Ecological Scarcity	104
D.9	ecoinvent database	104
D.10	Quicktips	104
	D.10.1 Pre or Post LCIA function	104
	D.10.2 Functions to update LCI data	105
D.11	Contact	106
E	Subversion Information	107

Chapter 1

What's new

1.1 Version 1.2

Version 1.2 has been created to support a new energy integration tool, called Eiampl. The tool aims to solve multi-period problems, where units and streams may be distributed in different locations, linked by distribution networks. Some new EI elements have been created and some existing (supported by Easy) are no more supported.

In this manual two new logos will appear to tell the user whether a particular EI element is supported by Easy, Eiampl or both. The two logos are the following:



Supported by Easy



Supported by Eiampl

Following section illustrates the major changes introduced in version 1.2.

1.1.1 EI entities

Layers A new *Layers* entity has been introduced by Eiampl in addition to the existing ones, like, Groups, Units or Streams. Layers represent a particular behavior, which consists in a MILP model. An example of layer is the heat cascade or a mass balance. More details are given in section [A.3](#).

Streams New stream types have been introduced in Eiampl:

- **Cost** (sec. [A.6.10](#))
- **vqt** (sec. [A.6.4](#))
- **vsteam** (sec. [A.6.8](#))
- **ht** (sec. [A.6.5](#))
- **mass** (sec. [A.6.11](#))
- **mq** (sec. [A.6.12](#))
- **mqpartial** (sec. [A.6.13](#))

Locations This new entity defines physical places where units and groups are located. Locations are isolated, which means that no resources can be exchanged between locations, unless a distribution network is defined. More information about locations is given in section [A.10](#).

Chapter 2

Introducing EnergyTechnologies

2.1 Introduction

This manual talks about EnergyTechnologies . Chapter 2 gives an overview of EnergyTechnologies and of the content of this manual. Chapter 3 defines the syntax of EnergyTechnologies . Chapter 4 details the syntax of additional interfaces allowing to enrich the generic model description. Finally, chapter 8 covers the syntax of the streams entering and leaving the generic model.

Chapter 5 covers the model validation procedure by defining the procedure itself and by explaining how to run it.

Chapter 7 covers the EnergyTechnologies Database , which is a database of models described with the EnergyTechnologies syntax.

The paper published by ? and presented at PSE conference 2009 in Brazil, gives a short overview of the reasons that lead to the creation of EnergyTechnologies .

2.2 What is EnergyTechnologies

EnergyTechnologies is a *markup language* intended to describe models developed in the domain of energy conversion systems. The goal of EnergyTechnologies is the definition of a structured language to be used to interrogate models in a uniform and generic way.

The definition of markup language given by Wikipedia¹ is the following:

A markup language is an artificial language using a set of annotations to text that describe how text is to be structured, laid out, or formatted.

In the context of EnergyTechnologies , the language defines a set of words (called *fields* later in the text) whose goal is to describe several aspects of a model. This includes the interfaces between the model and the external world, the files composing it, its documentation and other aspects, which will be documented in chapter 3.

A markup language is a descriptive language, which only defines a syntax. This differs from a complete programming language, where syntax is coupled with semantics, which defines the meaning of the syntax with rules and functions.

The EnergyTechnologies language is platform-, programming language- and software-independent. This means that the syntax can be used to describe models written with any software or programming

¹http://en.wikipedia.org/wiki/Markup_language, 23-jul-2008

language, each one having different ways of defining variables, parameters, inputs or outputs. To suggest a visual example, EnergyTechnologies is a sort of flexible envelope that fits the external boundary of a model. For this reason, it can adapt to each kind of model.

Semantics is provided by platforms like *OSMOSE* (?). *OSMOSE* uses EnergyTechnologies to connect models together and in general to query models by sending input parameters and recovering results. The *OSMOSE* manual (?) explains how to query a model and how to perform complex computations, like optimization. Chapter 7 also explain how to query a model for a simple evaluation.

2.3 Programming language

EnergyTechnologies is a programming syntax which can be implemented in many programming languages. In this document, all the examples are provided using Matlab language, since this software is used for other tools developed at LENI.

2.4 What is a model

A model is a set of equations and constraints which represents the behavior of an energy conversion technology and more generally, a combination of physical, thermodynamic and chemical phenomena.

A model is seen by EnergyTechnologies as a black box which is connected to the outside world through two kind of streams: material and energy. The internal behavior of the model can be accessed and controlled through the model's parameters. The concept is shown in figure 2.1.

A model can be as simple as a single equation or can represent several energy conversion technologies with hundreds streams and thousands parameters. It doesn't matter how much a model is complex, because EnergyTechnologies always sees it in the same manner: a black box with a certain number of streams crossing its boundaries and a certain number of internal parameters.

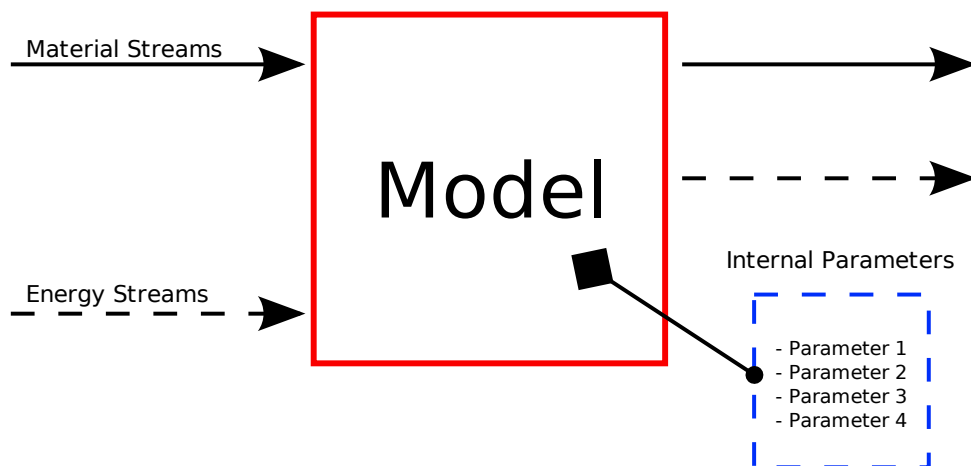


Figure 2.1: Illustration of EnergyTechnologies description of a generic model

EnergyTechnologies only describes the frontiers of the model and not the internals. This allows to provide a generic and well-structured interface to connect (very) different models (even built with different software or programming languages) and to allow communication between models and computation tools.

2.5 Extension of the generic model - Methods interfaces

In system analysis, there are several reasons to query a model, depending on the study which is performed. Each study is in general linked with a particular computation method which allows to obtain a series of indicators dependent of the model state. For example, the costing method allows to obtain some costing indicators (grass root, operating cost, ...) in function of the equipment size and resources consumption.

Some examples of computation methods widely used in system analysis are the following:

- Energy Integration and pinch analysis
- Costing
- Environmental Impact
- Life-cycle Analysis

EnergyTechnologies allows to extend the generic model description defined in section 2.4 by the mean of *method interfaces*. Each computation method requires an additional set of data to be included in the general model description. The syntax to be used to interface the model with computation methods is described in detail in chapter 3. Figure 2.2, which is an extension of figure 2.1 shows the concept of method interfaces.

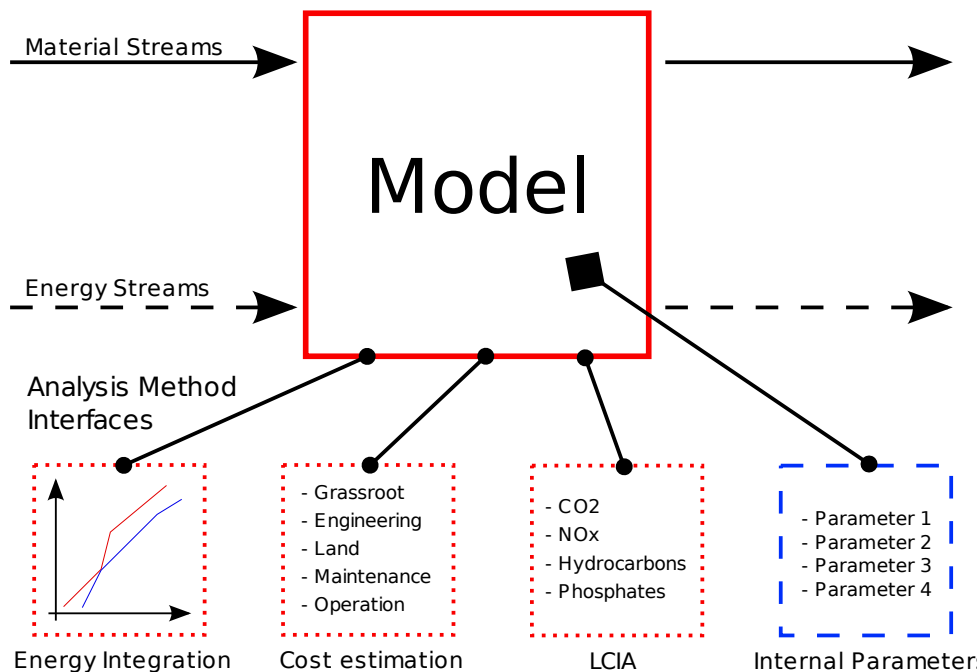


Figure 2.2: Illustration of EnergyTechnologies description of a generic model with methods interfaces

2.6 Programming strategy

The description of a model is programmed in EnergyTechnologies as a structure.

In computer programming, a structure is a particular type of variable which can be graphically represented as a tree. This variable type² is known by almost all programming languages (C, Matlab, PHP,

²The variable type defines in fact the kind of content that is accepted as value. To cite some examples, integer, boolean, floating number, string are variable types.

XML, ...). It allows to well organize code and is easy to translate form a language to another. Furthermore, structures can easily be converted into objects in languages which are object-oriented.

A structure has always a root, i.e. a parent element from where the whole structure is developed under a tree form. The root of the EnergyTechnologies structure is the generic model description. In the examples included in this document, the root will be represented by the `model` name. The description of the `model` content and of all of its sub-fields is given in chapter 3.

2.7 Running models

As explained in section 2.1, EnergyTechnologies is a description of the boundaries of a model. The model internals depend on the software or programming language used to build it. To query a model, the user has access to the parameters available in the model description. The transfer of information from the description to the real model is done by the *OSMOSE* platform (?).

OSMOSE is programmed to take advantage from the EnergyTechnologies syntax and is able to send and recover values to and from the model.

2.8 Models database - EnergyTechnologies Database

EnergyTechnologies is a syntax that allow to describe any model with a generic syntax. It appears then reasonable to group models to form a library. The EnergyTechnologies Database is that library.

Chapter 7 explains the working principles of the database and details how to use it.

Chapter 3

EnergyTechnologies syntax

In this chapter the syntax of EnergyTechnologies is described. The syntax is defined using a structured variable. The root (i.e. the main element) of the structure is the model, which is defined with the `model` field in the examples of this document.

The structured variable is chosen because it is available in almost all modern programming languages (like C, PHP, XML, ...) and is very easy to be translated from a language to another. For object-oriented languages, it is even possible to enrich the structure with classes and methods.

An example of structured variable is provided in listing 3.1.

```
% This is an example of structured variable
root = struct;
root.field1 = 'character string';
root.field2 = {'cell array','of strings'};
root.field3 = [2,1,4]; % a vector

root.field4 = struct; % a substructure
root.field4.subfield1 = 'another string';
```

Listing 3.1: Example of structured variable

Each field of the `model` structure is explained in the following paragraphs and some examples are provided. The description of a model is composed of several parts, each one describing a particular aspect. In particular:

Streams and parameters	Data about input and output streams and data about internal parameters of the model.
Interfaces	Data about interfaces.
Model files and compatibility	Information about model files and compatibility with <i>OSMOSE</i> and other software.
Grouping and classification	Provides information about technology classification, in order to establish links between models and resources.
Authoring	Data about model version, changelog, literature references, ...
Validation	Data about model validation status.

3.1 Variable types for fields

The content of each field of the EnergyTechnologies syntax belongs to a certain variable type. Tabular 3.1 resumes all the possible values that can be found in this document. Not all the programming languages allows to implement exactly the same variable type. In these cases, the most similar variable type will

be used.

Variable type	Description
<code>bool</code>	Boolean (1 0)
<code>int</code>	Integer variable
<code>float</code>	Real number
<code>string</code>	Characters string
<code>struct</code>	Structure
<code>mixed</code>	used when a field can contain several variable types
Container	Description
<code>ay</code>	Array. This is not really a variable type in EnergyTechnologies . It indicates a stack of several variables
<code>cell</code>	Cell. A cell is simply a container. Each cell can contain another variable type.
Specials	Description
<code>cell string *</code>	The content of the cell string is defined among a limited number of possible choiches. The field description gives a reference to the available values.
<code>int *</code>	The value of the integer variable has to be chosen among a limited number of choices.
Mixed Cases	Description
<code>cell string</code>	A cell containing a string
<code>ay struct</code>	An array of structures
<code>ay int</code>	An array of integers
<code>ay cell string</code>	An array of cells containing each a string
<code>ay float</code>	A matrix of real numbers
<code>ay cell mixed</code>	Array of cells containing several types of variables

Table 3.1: Variable types

3.2 Elements identifier - TagName

In this manual you will often find a field called `TagName`. This field is used to uniquely identify each element of syntax, for example streams, internal parameters or energy integration units. To give a name to each element, you have to respect some rules:

1. The `TagName` of each element must be unique. This means that no other element in the model can have the same name, even if it refers to something else (for example a `Stream` vs. a `Tag`).
2. The `TagName` must be start with a letter. No number or special characters are allowed.
3. No special characters are allowed in a `TagName`, except for `'_'` (underscore).
4. No spaces are allowed `TagName`.

Listing 3.2 shows some examples of valid and invalid `TagName`.

3.3 Streams and Internal Parameters

As shown in figure 2.1, the internal parameters allows to modify the behavior of the model, while the input and output streams allows to connect models together. This section provided the syntax to define input and output streams and internal parameters.

```

%% Valid code
TagName = {'abc'}
TagName = {'a243sdf'}
TagName = {'ghfdg34'}
TagName = {'sdgfgf_sfdg3'}
TagName = {'ASOUHLH'}
TagName = {'AcDnFm'}
TagName = {'a_a_a_'}

% Unvalid code
TagName = {'133254'} % starts with a number
TagName = {'5dgfdg'} % starts with a number
TagName = {'_43w55'} % starts with a special character
TagName = {'sdf@4'} % contains special characters
TagName = {':rf.='} % contains special characters
TagName = {'srdg set'} % contains spaces

```

Listing 3.2: Examples of usage of `TagName`

3.3.1 Material and Energy Streams

Material and energy streams define the connecting interface between models. Listing 3.3 shows the definition of a stream and provides some usage examples. In EnergyTechnologies 2.0, no information about quantity (ex. mass flow, power) or quality (ex. composition, pressure, temperature ...) is provided in the streams definition. The only information consists in the stream `Type`, which is an identifier pointing to an entry in the streams database (see sec. 8). This database will, in future versions of EnergyTechnologies, provide data and information about the stream.

3.3.2 Internal parameters

Internal parameters are the parameters that allows to control the model behavior. Internal parameters are accessible via the `model.Tags` structure, which is described in listing 3.4. Details about `Tags` fields are given in table 3.2.

Field	Var. type	Description
<code>Tags</code>	<code>ay struct</code>	Contains information about internal parameters
<code>TagName</code>	<code>cell string</code>	Allows to identify a parameter. Its name must be unique in the model.
<code>DisplayName</code>	<code>cell string</code>	Is used to display the name of the parameter in a pretty way.
<code>Description</code>	<code>cell string</code>	Defines in more details the meaning of the parameter.

Continued on next page ...

... Continuing from previous page

Field	Var. type	Description
Status	cell string	defines the status of the parameter; CST defines a fixed parameter, i.e. an input value. OFF defines a parameter which will be computed by the model, i.e. an output value. In some particular cases (which depend on the software used to build the model), numerical values are allowed to define the model status (ex. {'30%'}). In this case, a numerical value indicates an uncertainty around the average parameter Value and the parameter becomes an input and an output at the same time. Statuses ending with the percent sign (%) indicate relative uncertainty, while simple values indicate absolute tolerance.
DefaultValue	mixed	Indicates the default value of the parameter. This value will be used for input parameters, unless the user wants to modify it via computation constants or variables. Section 6.3 explains how to do that. The value may be numeric (float) or a string (cell string).
Unit	cell string	Indicated the measurement unit of the parameter, like m2, kg/s,
isVIT	int	Indicated whether the parameter is a <i>Very Important Tag</i> . In this case, the parameter will be displayed in graphical interfaces with major priority than other more obscure tags. Value 0 indicates that the tag is not VIT. Values larger than 0 indicates that the tag is VIT. The larger is the value, more the parameter is important. This feature can be used for sorting parameters for display purpose. Moreover <i>all</i> the parameters fixing a degree of freedom of the model <i>must</i> be defined as being isVIT . More information about this reason is given in section 6.2.
Value	float	Contains the value of the parameter once the model has been computed.

Table 3.2: Tags fields from listing 3.4

3.4 System synthesis and analysis methods interfaces

System synthesis and analysis methods interfaces (see fig. 2.2) allows one to enrich the model with additional information and data. Each interface attaches a substructure to the model structure, defining all the inputs required by the computation method. The results of the computation will also stored in that structure.

Each interface has its own syntax. When running a computation, the user must use the internal parameters of the model to modify the parameters of an interface. Section 6.4 will explain how to do that.

The syntax of the interfaces officially supported by EnergyTechnologies is covered in chapter 4.


```

model.Streams          % this structure contains information about streams entering or
    leaving the model.
model.Streams.TagName   % Stream identifier. Must be unique in your model
model.Streams.DisplayName % Pretty name to be shown in display
model.Streams.Description % Description of the stream
model.Streams.Type      % To be chosen in known types in streams database
model.Streams.InOut     % [{'in'|'out'}] Tells wheter the stream is entering or leaving
    the model
model.Streams.Params    % Reserved for future implemetations. Contains quality and
    quantity information
%
% No information about quantity or composition is provided at this stage.
%
% Example:
% model.Streams.TagName      = {'NatGas'};
% model.Streams.DisplayName  = {'Natural Gas'};
% model.Streams.Type        = {'methane'};
% model.Streams.InOut       = {'in'};
%
% or
%
% model.Streams.TagName      = {'elec_solar'};
% model.Streams.DisplayName  = {'Electricity from solar panels'};
% model.Streams.Type        = {'electricity'};
% model.Streams.InOut       = {'out'};

```

Listing 3.3: Matlab code for `model.Streams`

```

model.Tags = struct;      % This tructure contains all the internal parameters of the
    model.

model.Tags.TagName        % cellstr Short name of the varaible
model.Tags.DisplayName    % cellstr (optional) Pretty name of the variable
model.Tags.Description    % cellstr (optional) Description of the variable
model.Tags.Status         % cellstr [cst|var|int|off] Variable type
model.Tags.DefaultValue   % mixed Default value for the parameter. Will be used if the
    user doesn't specify any othervalue via the o.Variables or o.Constants structures
model.Tags.Unit           % cellstr (optional) Measurement unit
model.Tags.isVIT          % bool (optional) Is this tag a Very Important Tag? This defines
    tags which will appear on rich interfaces
%
% Read-only fields
model.Tags.Value          % float Is the value taken by the parameter. The value is
    computed by OSMOSE.
%
% Example:
% model.Tags.TagName      = {'Eff_elec'};
% model.Tags.DisplayName  = {'Electrical efficiency'};
% model.Tags.DefaultValue = 0.8;
% model.Tags.Unit        = {'-'};

```

Listing 3.4: Matlab code for `model.Tags`

3.5 Model files and links to other software

The definition of a model requires the specification of the number of files composing it and, for complex models composed by several files, the order which the files must be called. Listing 3.5 defines the fields necessary to define the model files. Table 3.3 details the content of these fields.

```

model.FileName          = {''};    % Name of all the files composing the model
model.Location          = {''};    % Absolute path indicating the folder where all the
    model files are stored
model.Software          = {''};    % indicates the name of the software used to build the
    main file(s) of the model
model.MainFile          = {''};    % Indicates which is the main file for the model, if more
    than one file is specified in FileName
model.PreModelMFunction = {''};    % Indicates the name of the matlab function to be called
    before running the MainFile in OSMOSE
% In the same style, also PostModelMFunction, PreEIMFunction,
% PostEIMFunction, .... are available. See OSMOSE manual for more details.

%% Identification
model.TagName           = {''};    % Unique name of the model
model.DisplayName       = {''};    % Pretty short name
model.Description       = {''};    % More detailed description than DisplayName. To be used
    to explain the model in some phrases
model.DescriptionImageFileName = {''}; % Name of the description picture

```

Listing 3.5: Definition of files composing a model

3.6 Version and compatibility

EnergyTechnologies and other software linked with it (like *OSMOSE*) may evolve during time. It is important to know which version of EnergyTechnologies has been used as reference when building a model description. The fields defined in listing 3.6 provide information about software versioning. They are detailed in table 3.4.

```

model.ETVersion         = {'2.1.0'}; % Indicates the version of
    EnergyTechnologies used to describe this model
model.OSMOSEVersion     = {''};      % Indicates the OSMOSE version to be used
    to run this model
model.Version           = {''};      % Indicates the version of the model. Should
    be updated for each changeset (see \ref{sec:authoring})

```

Listing 3.6: EnergyTechnologies code for model versioning

Field	Var. type	Description
<code>FileName</code>	<code>ay cell string</code>	Contains the name of all the files composing the model (including the model description)
<code>Location</code>	<code>cell string</code>	Indicated the full path where the model is stored.
<code>Software</code>	<code>cell string</code>	Indicates the name of the software used to build the mail file of the model. Only one software (plus Matlab) is allowed for each model.
<code>MainFile</code>	<code>cell string</code>	Indicated the main file of the model, if the model is composed by more than one file.
<code>PreModelMFunction</code>	<code>cell string</code>	This and other fields indicate the name of the matlab function to be called at each stage of computation. For more details, see the <i>OSMOSE</i> documentation (?).
<code>TagName</code>	<code>cell string</code>	This is the identifier of the model inside <i>OSMOSE</i> . Must be unique.
<code>DisplayName</code>	<code>cell string</code>	Pretty name to be used for displaying purposes.
<code>Description</code>	<code>cell string</code>	This field contains a short description of the model. More details are given in the <code>model.Description</code> fields (see sec. 3.9)
<code>DescriptionImageFileName</code>	<code>cell string</code>	Name of a picture file describing the model

Table 3.3: Description of model files fields defined in listing 3.5

Field	Var. type	Description
<code>ETVersion</code>	<code>cell string</code>	Indicates the version of EnergyTechnologies used to describe this model. Use the format X.Y.Z, like 2.0.0.
<code>OSMOSEVersion</code>	<code>cell string</code>	Indicates the version of <i>OSMOSE</i> used to run the model. Use the format X.Y.Z, like 2.0.0.
<code>Version</code>	<code>cell string</code>	Indicates the version of the model. Should be updated for each changeset (see 3.8.1).

Table 3.4: Description of versioning fields defined in listing 3.6

3.7 Model Classification

EnergyTechnologies Database , which is documented at chapter 7, requires information allowing to class models. Classification allows to organize and structure the information. EnergyTechnologies syntax allows to class models in several ways. In particular:

- By a static grouping based on a tree
- By a keywords system
- By streams compatibility connections

3.7.1 Grouping

Grouping provides a static general classification organized in tree with categories, subcategories and so on. Listing 3.7 gives the EnergyTechnologies syntax and table 3.5 details each field.

```

%% Classification by grouping
model.Group      = {''}; % First level of classification
model.Type       = {''}; % Second level of classification
model.SubType    = {''}; % Third level of classification
%
% Example
% model.Group     = {'CombinedCycle'};
% model.Type      = {'CogenerationWithDrawOff'};
% model.SubType   = {'HighTemperature'};

```

Listing 3.7: Classification by grouping

Field	Var. type	Description
Group	cell string	Defines the category which the technology belongs to. They are defined at subsection 7.1.
Type	cell string	Second level of classification.
SubType	cell string	When necessary, this field can be used to add additional information which cannot be included in the above fields.

Table 3.5: Grouping fields from listing 3.7

Classification tree

Technologies classification must follow well defined rules. They are defined in two matlab files: `ET_groups_list.m` and `ET_types_list.m`. The former defines the first level of classification. Its content is shown in table 3.6. The latter defines the second level of classification, described in table 3.7. The third level of classification, `subType` is not mandatory and the user can freely define it.

ID	Group	Description
1	BiomassConversion	Biomass conversion technologies
2	CombinedCycle	Combined cycles
3	Igcc	Integrated coal combined cycles
4	InternalCombustionEngine	Internal combustion engines
5	FuelCell	Fuel Cells
6	GasTurbine	Gas turbines
7	SolarCollector	Solar collectors
8	SteamTurbine	Steam turbines
9	Other	Reserved for technologies which cannot match any other category

Table 3.6: Classification groups

Gr. ID	Type	Description
1	FluidisedBedGT	Fluidized bed with Gas turbine
1	FluidisedBedICE	Fluidized bed with internal combustion engine
2	Cogeneration	Simple gas turbine with steam turbine
2	CogenerationWithDrawOff	Like Cogeneration , with district heating drawoff
3	Coal	Coal-based IGCC
4	Diesel	Diesel engine
5	MoltenCarbonate	Molten carbonate FC (MCFC)
5	PhosphoricAcid	Phosphoric acid FC (PAFC)
5	SolidOxyd	Solid Oxyd FC (SOFC)
6	Aeroderivative	Aeroderivative gas turbine
6	Heavy	Heavy gas turbine
7	Thermal	Thermal collectors
8	PulverizedCoal	Pulverized coal as fuel
9		No type can be defined

Table 3.7: Classification Types

3.7.2 Keywords

Keywords (also known as tags. EnergyTechnologies uses keywords to avoid misunderstandings with model's internal parameters, called `model.Tags`) allow to attach descriptions to a model without following a predefined schema. This mode of classification is more flexible than the grouping classification, because several keywords can be associated to each model. Each keyword allows to class the model with different criteria, while the grouping classification only allows one, the group-type-subtype based on technologies typologies.

EnergyTechnologies uses triple tags technique. This is the triple tags definition given on Wikipedia¹:

A triple tag is a tag that uses a special syntax to define extra information about the tag, making it easier or more meaningful for interpretation by a computer program. Triple tags comprise three parts: a *namespace*, a *predicate* and a *value*. For example, "geo:long=123.456" is tag for the geographical longitude coordinate whose value is 123.456.

Using triple tags has two main advantages:

1. The predicate allows to better specify the context of the tag and can be used to create a classification category.
2. The namespace allows to isolate tags from an application to another. EnergyTechnologies being a description syntax, many software may use it. Each software can define its own namespace and can classify the models with its own criteria.

Tags defined in EnergyTechnologies for models classification use the `et` namespace. Listing 3.8 shows the syntax of EnergyTechnologies keywords and table 3.8 gives more details.

Field	Var. type	Description
Keywords	ay cell string	Defines classification keywords using triple tags technique.

Table 3.8: Classification by keywords. See listing 3.8

3.7.3 Stream Connections

Input and output streams declare the way the model interacts with its environment. Two models having respectively an output and an input stream of the same type (`model.Streams.Type`) are compatible and

¹Taken from http://en.wikipedia.org/wiki/Machine_tag at July 17, 2008

```

%% Classification by keywords
model.Keywords = {''; % Contains a vector of keywords to be used to
    class the model. Accepts machine tags.
%
% Examples
% model.Keywords(1) = {'namespace:predicate=value'};
% model.Keywords(2) = {'et:group=CombinedCycle'};
% model.Keywords(3) = {'et:type=CogenerationWithDrawOff'};

```

Listing 3.8: Keywords syntax

could be connected together.

A superstructure of models is in fact a sort of classification, where the links of the tree are defines by material or energy streams.

3.7.4 Modeling detail

Each model is developed with the goal to represent the reality with a certain precision. Some models will be developed to quickly provide results, but their precision will be moderate. Other models will represent the reality in the finest detail, including 3d geometries and time-dependent phenomena.

The `model.PhysicalRepresentation` field is there to inform the user on the detail level of the model. The accepted details levels are defined in the `ET_PhysicalRepresentation_list.m` function and are resumed in table 3.9.

Physical representation	Description
<code>constant</code>	Models in this family only provide constant values.
<code>linear</code>	These models are suitable to be used with MILP solvers. Results are linearly proportional to a reference value.
<code>nonlinear</code>	These models are built with nonlinear equations.
<code>0d</code>	May be NLP models. No geometrical dimension is taken into account.
<code>1d</code>	May be NLP models. Only one dimension is taken into account.
<code>2d</code>	Two geometrical dimension are taken into account.
<code>3d</code>	3-dimensional geometries are taken into account.
<code>steadyState</code>	Models are not time-dependent.
<code>multiPeriod</code>	Models are time-dependent. Each time step is associated with a particular state in a discrete way.
<code>dynamic</code>	Models are time-dependent. In this family, equation are analytical and can be differentiated.

Table 3.9: Classification groups

The `model.PhysicalRepresentation` may contain several physical representations. An example is given in listing 3.9.

3.8 Authoring and Bibliography

A model must be documented and validated to be trusted. EnergyTechnologies provides several fields to store documentation, authors identity, changelogs, bibliographic references and other information linked with model history.

```

model.PhysicalRepresentation = {''};      % Provides information about the level of
      detail of the model.
%
% Example:
%   model.PhysicalRepresentation = {'1d'};
%   model.PhysicalRepresentation = {'2d','dynamic'};
%   model.PhysicalRepresentation = {'linear'};

```

Listing 3.9: Physical representation example

3.8.1 Authoring

A model may evolve during time. Each change may be incompatible with the previous version. For this reason EnergyTechnologies has a `model.Version` field (see sec. 3.6). In addition to this field, EnergyTechnologies also proposes more details about changes. The `model.Changesets` structure allows to reference all changes including author name, date and bibliographic references. Listing 3.10 shows the syntax of that structure, while table 3.10 provides details about its fields.

```

%% Technology authoring
model.Changesets = struct;
model.Changesets.Id = [];          % Changeset identifier (integer)
model.Changesets.Author = {''};    % Author's name
model.Changesets.Date = {''};      % Date of changeset
model.Changesets.ChangeLog = {''}; % Text describing the changes
model.Changesets.ReferenceId = {''}; % Link to references (same value as model.
      References.Id)
%
% Example
% model.Changesets.Id = [1];
% model.Changesets.Author = {'Raffaele Bolliger'};
% model.Changesets.Date = {'17-jul-2008'};
% model.Changesets.ChangeLog = {'Initial release'};
% model.Changesets.ReferenceId = {''};

```

Listing 3.10: Changesets fields

Field	Var. type	Description
Changesets	ay struct	Structure containing all changesets
Id	int	Unique identifier
Author	cell string	Name of the author
Date	cell string	Date of the changeset. To be expressed as dd-mmm-yyyy.
ChangeLog	ay cell string	List of changes. Listing one change per cell
ReferenceId	ay int	List of reference identifiers (see sec. 3.8.2)

Table 3.10: Changesets fields from listing 3.10

3.8.2 Bibliographic references

A good model has bibliographic references, which may be from books, papers, web sites, technical catalogs, experimental measurements and so on. EnergyTechnologies provides fields to account for references. They are detailed in listing 3.11 and table 3.11. Reference sources must be reported using BibTeX syntax.

```

%% Bibliographic references
model.References          = struct;
model.References.Id        = [];    % Reference Identifier (integer)
model.References.SourceFileName = {''}; % Name of the file containing the reference. Use
                                     BibTeX style
model.References.BibTexId   = {''}; % Keyword allowing to indentify a BibTex entry.
model.References.Comments   = {''}; % Comments

%% Example
model.References          = struct;
model.References.Id        = [1];
model.References.SourceFileName = {'biblio.bib'};
model.References.BibTexId   = {'osmose_doc'};
model.References.Comments   = {'OSMOSE Documentation'};

% This is from biblio.bib
% ...
% @TechReport{osmose_doc,
%   abstract = {Official OSMOSE Documentation},
%   address  = {LENI-STI-EPFL, Batiment ME.A2.434, Station 9 CH-1015
%               Lausanne, Switzerland},
%   author   = {Palazzi, Francesca and Perin-Levasseur, Zoe and
%               Bolliger, Raffaele and Gassner, Martin},
%   title    = {OSMOSE User Manual},
%   institution = {LENI-STI-EPFL},
%   year     = 2007
% }
% ...

```

Listing 3.11: Bibliographic references

Field	Var. type	Description
References	ay struct	Structure containing all references
Id	int	Unique identifier
Source	cell string	Reference. Use BibTeX syntax.
Comments	cell string	User comments.

Table 3.11: Bibliographic references from listing 3.11

3.9 Documentation

Documentation is probably as important as the model programming quality. Without documentation, a model is difficult to understand.

EnergyTechnologies provides a `model.Documentation` structure holding all the parameters necessary to access or build the documentation of the model. Two strategies are available:

1. The documentation comes with the model, with a PDF file, the sources, the images and everything necessary to build it. In this case, the `model.Documentation` field provides the names of the bundled files.
2. The documentation is sorted remotely, on a web server. The `model.Documentation` provided the `Url` of the remote server where the documentation is available. `prewtcg`

Listing 3.12 and table 3.12 detail the syntax.


```

%% Documentation
model.Documentation          = struct;
% Remote documentation
model.Documentation.Url      = {''}; % Remote address where doc is displayed.
% Embedded documentation
model.Documentation.ReleaseFileNames = {''}; % Names of the files to be distributed (ex.
pdf)
model.Documentation.SourceFileNames = {''}; % All source files to included in
documentation
model.Documentation.MainFileName  = {''}; % Name of the main source file

```

Listing 3.12: Documentation fields

Field	Var. type	Description
Documentation	struct	Structure defining all documentation files
Url	cell string	Web url pointing to the documentation page
ReleaseFileNames	ay cell string	Name of the files to be released. In general, here are the compiled sources.
SourceFileNames	ay cell string	Name of the source filed for building the documentation.
MainFileName	cell string	Name of the main file to be called from sources to build the documentation

Table 3.12: Documentation fields from listing 3.12

3.10 Validation

A model can be trusted and distributed when it has achieved a certain number of revisions. A large number of bibliographic sources is a good indicator of the model quality.

EnergyTechnologies uses several indicators to establish the overall quality of a model:

- Syntax validation
- Simple model execution
- Number of bibliographic sources
- Number of measurements (or experimental data, ...) used to calibrate the model
- Peer review

The validation procedure, covered in chapter 5, establish a rating of the model. The higher is the value (on a maximum of 100), the better is the model.

The fields of the `model.Validation` structure are illustrated in listing 3.13 and in table 3.13.

```

%% Model validation
model.Validation          = struct;      % Contains all the validation fields
model.Validation.Status   = {''};       % Validation stage: toDo, Valid, toUpdate,
    inProgress --> available states are defined in ET_validationStatus_list.m
model.Validation.Comments = {''};       % Any comment about validation can be put here
model.Validation.Rating   = {''};       % Provides a rating for the model.
model.Validation.NbUse    = {''};       % Indicates the number of users having
    satisfactory obtained results whith the model.

```

Listing 3.13: Validation syntax

Field	Var. type	Description
Validation	struct	Structure containing validation data
Status	cell string	Indicates the status of the actual model validation. This field can either be manually modified by the user or automatically changed by the model validator (see. chap. 5), if a change in the model is identified. Possible statuses are described in table 5.1.
Comments	cell string	A comment can be left here to explain the model validation status.
Rating	int	The rating is computed by the validation procedure and indicated the quality of the model. The value is spread from 0 to 100.
NbUse	int	When the model has reached the final status (rating = 100), the number of users having satisfactory used the model is stored in this parameter. That gives a second quality indicator.

Table 3.13: Validation syntax 3.13

Chapter 4

Method Interfaces Syntax

4.1 Why Method Interfaces?

The generic model view explained in section 2.4 defines a model as a black box. The only knowledge about it is given by the declaration of the input and output streams and by the access to the internal parameters. This vision is very useful, because it allows to describe many different models with a uniform syntax. Moreover, this technique creates an interface allowing to connect models and to create super-structures, which is one of the goals of system analysis.

However, a simple black box and some streams don't tell us much about the model and discourage the development of very accurate representations of the reality, since the fine detail is lost inside the box. Furthermore, system analysis is a research domain where many different disciplines meet. Each one proposes a computation method providing a certain number of indicators based on the model behavior. Some examples of computation methods are resumed in the following list:

Method	Description
Energy integration	Computes energy savings and allows to design the heat exchange network.
Costing	Provides information about technologies cost.
Environmental impact	Considers pollutants and substances having an impact on the environment.
Life Cycle Analysis	Computes the emissions released during the whole life cycle of a model.

Method interfaces allows to enrich the basic model definition with additional information provided by computation methods. A model can then be used for several kinds of studies, depending on the interests of the researcher. This modelling layout is extensible and new method interfaces can be added without requiring to rewrite the model.

4.2 How do they work?

Figure 2.2 shows that method interfaces are separated from the general model definition. Each method has its own structure, where all the information necessary to run the method are stored. After computation, the results concerning the model are also stored in the same structure.

The separation of the information between the model and the method interfaces is important for many reasons. First, each method is isolated from the others to avoid collision between variable names or data. This allows to develop methods independently from the others. Then, if a method is improved, only its structure must be upgraded, while the remaining model may left untouched.

In the examples given in the annexes, fields values are sometimes replaced by strings. This is a technique allowing to dynamically change the value of the field in function of the model state. This is

done by replacing the field value with the name of an internal parameter of the model (see sec. 3.3.2). More details concerning this technique are given at section 6.4.

4.3 Officially supported method interfaces

The list of officially supported method interfaces bundled with EnergyTechnologies and *OSMOSE* is resumed in table 4.1. The syntax of these interfaces is given in annexes.

Interface	Structure	Annex	Description
Energy Integration	<code>model.EI</code>	A	Computed the energy integration of the model. Requires the definition of hot and cold streams.

Table 4.1: Officially supported method interfaces

Chapter 5

Model Validation

The validation of a model is a process aiming to check the compatibility of the model with the quality standards required by EnergyTechnologies . Standards concern following domain:

- Model description coding
- Bibliographical sources
- Model robustness and validity range
- Model calibration on measurements and/or other models
- Peer review

Part of the validation can be performed automatically, like checking the model description or testing the robustness. Other validation tasks must be performed by hand.

This chapter explains the various validation stages and defines the quality standards to be met in order to consider a model fully valid.

5.1 Validation status

The validation status of a model is described with the `model.Validation.Status`. The function `ET_validationStatus_list.m` defines the accepted statuses. Table 5.1 reports them.

Status	Description
ToDo	Validation didn't happen. Someone has proposed the development of a new model.
Specified	The parameters are defined and classified. The input and output of the model are clearly determined.
Valid	The model computation has been checked. The computed values are coherent.
UserValid	At least one user had satisfactory results using the module. The user can not be the developer.
ToUpdate	Someone found one or more problems in the model. Model validation must be performed again.
InRevision	Someone is updating the model. Nobody should modify it.

Table 5.1: Validation statuses

5.2 Rating procedure

Assuming the same weight for each validation steps, the rate is given as a fonction of the number of validation satatus fulfilled minus one compared to the total number of validation status to be done minus one. The validation status ToUpdate and InRevision are not considered in this total. Therefore, the first status ToDo corresponds to the rate value 0 and the last one to the rate value 100.

5.3 Model Description

The validation of the model descrption, defined with the EnergyTechnologies syntax (see chap. 3) is performed with a computer routine. The validation checks that the necessary fields are filled and that the information is correctly set.

Each method interface should have its own syntax validation routine, which can be integrated in the main model syntax validation.

5.4 Bibliographical sources

The number of bibliographical sources is an indicator of the model quality. The higher is the umber of sources, the better should be the quality of the model. Nevertheles, this is not always true. For this reason the appreciation of the model quality is left to the peer reviewer (see sec. 5.7).

The validation of the bibliographical sources can be partially automated. The following points are verified:

- The completeness of the bilbiographical sources (dates, titles, authors, ...).
- The number of sources.
- Links with changesets (see sec. 3.8.1).
- TODO

5.5 Model robustness and validity range

Automatic tests are performed to check if the model can be run in nominal conditions (i.e. with internal parameters set with default values, see sec. 3.3). If the model works, other tests can be run with a variable range of parameters, in order to verify or detect the model validity range.

TODO

5.6 Model calibration on measurements and/or other models

Automatic tests are run to verify the fitness of the model on measurements or data from other models.

TODO

5.7 Peer review

Peer review allow to ...

TODO

Chapter 6

Model Usage

This chapter explains how to run a model and how to interact with it by changing the value of the internal parameters or the parameters of the method interfaces.

6.1 *OSMOSE* - Running the models

OSMOSE is a computation platform developed under Matlab. Its goal is to link models and computation methods, whatever are the software used to build them.

OSMOSE implements the EnergyTechnologies syntax to handle models and is able to communicate with them by sending or recovering data. The way this is done is explained later, in sections 6.3 and 6.4.

The *OSMOSE* documentation (?) explains in detail how to run complex computations, like optimization or sensitivity analysis. In this chapter, only the basics are given. To be more precise, here will be explained how to run a model to compute one state (in *OSMOSE* this is called *OneRun*) in function of internal parameters values (sec. 6.2 and 6.3).

6.2 Run a model in its default state

A model correctly described with the EnergyTechnologies syntax contains all the information necessary to be run. The internal parameters have a *DefaultValue* (see sec. 3.3.2) field defining the default value of each parameter. Moreover, the *isVIT* field defines if the parameter is important for the model. *isVIT* tags include all the degrees of freedom of the model.

To compute the default state of the model, *OSMOSE* sends the default value of all *isVIT* parameters to the model and recovers the computation results.

Data are sent to *OSMOSE* using a structured variable, called *o*. It contains many information, including computation parameters, software paths and computation results. Information about the model is stored in *o.Model*. There you can find the whole model description defined in section 3 and other fields which may appear during the computation.

The *OSMOSE* documentation explains how to configure a computation using a Matlab script called *frontend*. This file is very detailed and allows to configure the computation with many options. EnergyTechnologies offers a simplified method allowing to run a *OneRun*. To do that, you have to define one variable containing the name of the matlab file describing the model and call the function (*ET_runModel*) that does the rest of the work for you.

Listing 6.1 shows an example of this procedure. A benchmark model called *EI_problem_1s* is computed with *osmose*.

After the computation has run, *OSMOSE* sends back the computation results, which are included in the *o* structure. At this stage, two fields are interesting: *o.Convergence* and *o.Results*. The former tells

```

%% Defining the variable containing the name of the function describing the model
o.ET.ModelMFileName = {
    'EI_problem_1s',...
};

%% Calling ET_runModel, which loads all the parameters to run OSMOSE
o = ET_runModel(o);

```

Listing 6.1: Running a model with default values

you if the computation converged (in this case, its value is 1). The latter contains the results of the computation. The values of all the internal parameters are stored there, together with the values of all the method interfaces. The *OSMOSE* documentation gives more details about the content of `o.Results`.

To recover an internal parameter, you can use the `osmose_results_getTag` function, as shown in listing 6.2. The `tag` contains the internal parameter structure defined in section 3.3.2.

```

%% Recovering tag 'h1_dt' from model 'EI_1s'
tag = osmose_results_getTag(o,'h1_dt','EI_1s')

```

Listing 6.2: Example: how to recover an internal parameter after computation

To recover all the internal parameters at once, it is possible to use the `osmose_results_getAllTags` function, as shown in listing 6.3. Note that in this case, all tags for all models are recovered. *OSMOSE* allows to run several models at once. An additional field in the `tags` structure (`ModelTagName`) contains the name of the model linked with the parameter.

```

%% Recovering all tags from all models
tags = osmose_results_getAllTags(o)

```

Listing 6.3: Example: how to recover all the internal parameters after computation

6.3 Modify model parameters

Using *OSMOSE*, it is possible to run a model by sending internal parameters values different than defaults. *OSMOSE* uses two similar structures to change the state of the model's parameters, called `o.Variables` and `o.Constants`. The difference between the two structures depends on the type of computation run with *OSMOSE*. For further details, see the *OSMOSE* manual. In the case of the `OneRun` computation both `o.Variables` and `o.Constants` are almost equivalent. In other cases, `o.Constants` holds parameter values which will remain static during the whole computation, while `o.Variables` contains the definition of parameters which are variables during the computation.

A difference between variables and constants still persists in `OneRun` computations. It concerns multi-period models. Only one value per Variable is defined and is used for all the periods, while one value per period can be defined for constants. This is an advanced topic and is not covered in this documentation. *OSMOSE* documentation provides more details.

For the `OneRun` computation, the definition of `o.Variables` and `o.Constants` is defined in listing 6.4. Apart the field name, the structure of both structures is the same.


```

o.Variables.TagName      % Short name of the variable, the same as the parameter that
                        must be changed
o.Variables.DisplayName  % (optional) Pretty name of the variable
o.Variables.Value        % Variable value
o.Variable.ModelTagName  % Name of the model which the parameter behaves. In our case,
                        use model.TagName

```

Listing 6.4: Matlab code for `o.Variables` and `o.Constants`

Listing 6.5 repeats the example of listing 6.1 and sends a new value for the `h1_dt` paramter.

```

%% Defining the variable containing the name of the function describing the model
o.ET.ModelMFileName = {
    'EI_problem_1s',...
};

% Initialization of variables counter
nv = 0;

nv = nv+1;
o.Variables(nv).TagName      = {'h1_dt'};
o.Variable(nv).ModelTagName  = {'EI_1s'};
o.Variables(nv).Value        = 15;

%% Calling ET_runModel
o = ET_runModel(o);

```

Listing 6.5: Running a model and changing the value of an internal parameter

6.4 Modify Method interfaces parameters

The `model` structure contains many data. Model's internal parameters are hold in `Tags` structure and data for method interfaces are hold in their respective structures (`EI` for energy integration, for example). This information is directly accessible in *OSMOSE* by browsing inside the `o.Model` structure.

To avoid programming problems, *OSMOSE* allows only one door to enter values in a Model: the internal parameters. This is done by defining variables or constants, like shown in section 6.3.

But, how can one modify method interfaces' parameters? How to modify for example the ΔT_{min} of a `qt` stream? This is possible by linking `Tags` with method interfaces parameters. Instead of setting a value for a parameter, you can put the `TagName` of a `Tag` preceded by a `@` sign. The `@` sign tells *OSMOSE* to recover the value from a `Tag`.

Listing 6.6 shows that the `'h1_dt'` used in the example of listing 6.5 was in reality a link to the ΔT_{min} of stream `'h1'`. The same thing is done for parameters `'h1_tin'`, `'h1_tout'` and `'h1_h'`.

The syntax of Tag links a powerful tool able to access all the `Tags` produced by the models and by *OSMOSE*. The simplified syntax shown in the prevoius paragraphs shows how to access `Tags` within the same model. Listing 6.7 show the complete syntax allowing to acces tags from other models and from analysis methods interfaces.

Namespaces are used to define where to search for a given Tag. Several namespaces are available:

```

%% in EI_1s model description

% ....

% EI Streams
ns = ns+1;
model.EI.Streams(ns).Short = {'qt','hot','h1','@h1_tin',100,'@h1_tout',0,'@h1_dt','@h1_h'};

% Tags
nt = nt+1;
model.Tags(nt).TagName      = {'h1_dt'};
model.Tags(nt).DisplayName  = {'DT/2 of qt stream h1'};
model.Tags(nt).DefaultValue = 10;
model.Tags(nt).Unit         = {'DK'};

% ...

```

Listing 6.6: Running a model and changing the value of a parameter of EI method interface

Model Tag link points to a Tag defined in a model

EI Tag link points to a Tag defined in EI interfaces

LCIA Tag link points to a Tag defined in LCIA interfaces

```

%% Syntax of Tags links

%% Links to Tags within the same model
@TagName

% Example
@Power_2 % links to tag "Power_2"

%% Links to Tags in other models
@ModelTagName.TagName % notice the dot (".") to separate model and tag names

% Example
@Model2.Power_2 % links to tag "Power_2" in model which TagName is "Model2"

%% Use of namespaces
@BaseStruct:ModelTagName.TagName % Notice the colon (":") to separate the namespace from
the tag reference

% Examples
@Model:Model2.Power_2 % same as @Model2.Power_2. Osmose assumes by default "Model" as
namespace
@EI:HTX_NMINMER % Models are not defined in EI interface, so ModelTagName is not required
here

```

Listing 6.7: Syntax of Tags link

Chapter 7

EnergyTechnologies Database

The EnergyTechnologies syntax defined in chapter 3 allows to describe models in a generic way. The same type of description can be applied to very different models. The *OSMOSE* computation platform is able to communicate with number of software and is designed to run models built with the EnergyTechnologies syntax and the relative system synthesis and analysis methods.

The creation of a database of models is a natural consequence and offers many advantages:

- Centralized place where to get models
 - Easy mintenance and upgrade
 - Models improved by shared knowledge
 - Functions allowing to browse the databse and to find a technology
- ...and many other.

This chapter illustrates the content of the database and explains how to use it.

7.1 Implemented technologies

TODO: update this section

The following list shows the different technologies implemented in the database. The field **Group** is used in the `model` structure to identify them.

- Internal combustion engines (Natural Gas, diesel, dual fuels)
- Gas Combined-cycles
- Gas Turbines (Aeroderivative, heavy)
- Integrated Gasification Combined-Cycles (Coal, Syngas)
- Pulverized coal combined-cycles (Subcritical, Supercritical)
- Fuel Cells (Molten carbonate, Phosphoric-acid, solid-oxyde, PEM)
- Fuel cell - gas turbine combined-cycles
- Heat pumps (Absorption, Compression)
- Boilers (Natural Gas, Light Fuel Oil, Biomass)
- Biomass conversion into power or syngas

- Distribution Networks (Water and CO_2 district heating)
- Tanks (Domestic Hot Water, Heating Hot Water, Ice)
- Organic Rankine Cycles
- Cooling cycles (Aerocondensers)
- Solar technologies (Thermal, Photovoltaic)

7.2 Database Usage

7.2.1 Technologies Identification

TODO

7.2.2 Technology search with classification method - `identifyMFielName`

In section 7.2.1, we explained how it is possible to search a particular technology by specifying at least one of the following fields:

- `model.Group`
- `model.Type`
- `model.SubType`

The `identifyMFielName` function allows to perform such search. The result of the search is the following.

Nothing found In this case a warning message tells that no technology has been found with the search inputs.

One technology found In this case the output of the function is the name of the Matlab function associated with the queried model.

More than one technology found In this case the function tells the user that more than one technology has been found and prints a list of all the items found.

Listing 7.1 shows an example of usage of the `identifyMFielName` function.

```
% Writing Inputs
model.Group = 'GasTurbine';
model.Type = 'Aeroderivative';

mFileName = identifyMFielName(technology);
```

Listing 7.1: Example of usage of the `identifyMFielName` function

7.2.3 Information about the database - `EnergyTechnologiesInfo`

TODO: does this function still exist?

This function provides information about the database. In particular:

- Database revision: on linux systems, if the Subversion revisioning system is installed, the maximum revision of the database is provided
- Paths: the function provides pathnames of the database itself and of all the subfolders

Listing 7.2 shows the function usage.

```
% 'info' is a structured variable containing, among others:  
% info.MaxRevision  
% info.BasePath  
% info.dbPath  
  
info = EnergyTechnologiesInfo;
```

Listing 7.2: Usage of the `EnergyTechnologiesInfo` function.

7.2.4 Browse the database - `browse_db`

TODO: does tis function still exists?

This function allows to browse the database to find all the available technologies. Each function in the "db" folder is run and if the output is a structure and it contains the `Group` field, then it is added to the available technologies vector `availableTech`. Listing 7.3 shows the usage of this function.

```
% 'availableTech' is a vector containing all the fields recoverable from the database  
% technologies.  
  
availableTech = browse_db;
```

Listing 7.3: Matlab code for the `browse_db` function

Chapter 8

Input and Output Streams Database

The whole chapter has to be written. Following text is simply a draft of ideas.

TODO

- goals / scope
- actual situation
- planned developments

8.1 Basic syntax

TODO

Lists the basic fields used to define a stream, like the oned defined in listing [3.3](#).

8.2 Extended syntax

TODO

Describe syntax depending on stream category. For example:

- Composition
- Thermodynamic state
- Fuel properties
- Method interfaces requirements (emissions, envrionmental impact)

8.3 Streams types

Defines streams categories, like fuels or pollutants, A stream can belong to several categories. Method interfaces can request stream types to be defined in order to run the model (or is it too complicated to implement?)

8.3.1 Fuels

The database reports the fuel type (like Natural Gas, Light Fuel Oil, . . .) and some other information about composition and chemical properties:

Fuel types The following fuels are available:

- No Fuel, `none`
- Natural Gas, `natural_gas`
- Coal, `coal`
- Wet Wood, `wet_wood`
- Dry Wood, `dry_wood`
- Diesel, `diesel`
- Gasoline, `gasoline`
- Hydrogen, `hydrogen`
- Propane, `propane`
- Kerosene, `kerosene`
- Heavy Fuel Oil, `heavy_fuel`
- Light Fuel Oil, `light_fuel`

Fuel State The following states are defined for fuels:

- Gas (`gas`)
- Liquid (`liquid`)
- Pulverized (`pulverized`)
- Chip (`chip`)

These states are used to define some fuels categories. The fuel state is defined in the `Fuel` structure as `State`.

Chemical Properties

Humidity, `Humidity`, [kg_{H_2O}/kg_{Fuel}] When applicable, humidity of solid fuels.

Lower Heating Value, `LHV`, [kJ/kg_{dry}]

Higher Heating Value, `HHV`, [kJ/kg_{dry}]

Exergy Heating Value `Exergy`, [kJ/kg_{dry}]

Adiabatic temperature `Tad`, [K] Temperature of fuel combustion in stoichiometric conditions.

Fumes mass-flow ratio `Fumes`, [-] Ratio between fumes mass-flow and fuel.

Air to fuel ratio `AirToFuel`, [-] Mass ratio between air mass-flow and dry fuel at stoichiometric conditions.

Fuel consumption `Consumption`, [kg/s] Fuel consumption mass flow.

Composition Following compounds are considered in the fuel composition

- Carbon **C**, [%]
- Hydrogen **H**, [%]
- Oxygen **O**, [%]
- Sulfur **S**, [%]
- Nitrogen **N**, [%]

Matlab code for Fuels The code in listing 8.1 is the Matlab output relative to the fuel properties. This code is a sub-structure of the main **technology** structure. It can be called by typing **model.Fuel**.

```
to rewrite. This code is not valid

model.Fuel.Type           % string      : Fuel identifier
model.Fuel.State          % string      : gas,liquid, pulverized, chip
model.Fuel.Humidity       % [-]        : humidity of solid fuels
model.Fuel.LHV            % [kJ/kg_dry] : Fuel Lower Heating Value
model.Fuel.HHV            % [kJ/kg_dry] : Fuel Higher Heating Value
model.Fuel.Exergy         % [kJ/kg_dry] : Fuel Exergy Heating Value
model.Fuel.C              % [-]        : Carbon composition
model.Fuel.H              % [-]        : Hydrogen composition
model.Fuel.O              % [-]        : Oxygen composition
model.Fuel.S              % [-]        : Sulfur composition
model.Fuel.N              % [-]        : Nitrogen composition
model.Fuel.Tad            % [K]         : Adiabatic combustion temperature
model.Fuel.Fumes          % [Kg/Kg_dry] : Massoflow ratio between fumes and fuel
model.Fuel.AirToFuel      % [Kg/Kg_dry] : flowmass ration between air and fuel
model.Fuel.Consumption    % [Kg/s]       : Fumes consumption mass flow
```

Listing 8.1: Fuel structure

8.4 Streams database

Lisits the officially supported streams



Appendix A

Process Integration

A.1 What is energy integration?

Energy integration, also known as pinch analysis, is a technique originally applied in chemical processes design to reduce energy consumption by maximising heat recovery between operating units. The technique has nowadays been applied to all the domains where production units need to exchange heat both for cooling or heating purposes.

The basic principle of the method is based on the definition of hot and cold streams, the former needing to be heated and the latter needing to be cooled. This is done for each operating unit in a production process. The analysis of the requirements of the streams allows to identify the possible exchange compatibilities by heat recovery. The result is a reduction of the process requirements of energy supplied from an external source.

Two softwares are available for solving energy integration problems: Easy and Eiampl. The various EI elements described in the following chapter are not always supported by both software. For this reason, two logos are defined to indicate which software supports it: Easy () and/or Eiampl ()

Eiampl introduces new features allowing to perform a complete process integration, including heat and mass resources. New concepts like different location and resource distribution networks are also introduced.

A.2 System methods interface syntax

The method requires both streams and units to be defined in order to run a computation. In addition, equations can be added to define constraints between streams or units.

The energy integration interface uses the `model.EI` structure. The first-level fields of the structure are shown in listing A.1. Details about the content are given in the following paragraphs.

The syntax of the `model.EI` structure was originally based on the implementation of the software used for compute energy integration, Easy. Its manual is available at:

 http://leni.epfl.ch/exsys/EASY_On_line_manuals/
 <http://leniwiki.epfl.ch/leniwiki/index.php/Easy>

```

model.EI.Layers           % Layers definition
model.EI.Units            % Units definition
model.EI.Groups           % Groups definition
model.EI.Streams          % Streams definition
model.EI.Equations        % Equations definition
model.EI.EqTerms          % Equation terms definition
model.EI.Connections      % Deficition of heat exchange connections
model.EI.Variables        % Defines variables to be used for energy integration
model.EI.Locations        % Defines locations where units and grous are situated.

```

Listing A.1: Content of the `model.EI` structure

A.2.1 Hierarchy - Containers and content

The syntax of the energy integration interface defines two kinds of entities: containers and content. `Groups`, `Units` and `Equations` are containers. While `Streams`, `Variables` and `EqTerms` define the content. `Units` may contain `Streams`, `Equations` contain `EqTerms`, while `Groups` may contain `Groups`, `Units`, `Variables` and `Equations`.

In the energy integration interface syntax, the containment hierarchy is built from bottom to top, from content to containers. This is done with the `Parent` field. So, for example, a `Stream` will have a `Unit` as a parent, which could be part of a `Group`.

The top entity is the main group, which is created by default by the interface. To insert an entity inside the main group, you can define `model.EI(...).Parent = 'main'`.

`Connections` are special in this sense, since they are not containers or content.

`Layers` are nor hierarchically organized, i.e. they are all at the same level. For this reason, the `Parent` field is not defined for these elements.

A.2.2 Generic fields for all the syntax elements

Supported by:  and 

Fields reported in listing A.2 and table A.1 are common to all container and content entities (and even for connections): `Units`, `Groups`, `Equations`, `streams`, `EqTerms` and `Connections`. They allow to uniquely identify the entity, to provide information about inheritance (`Parent`) and to configure the usage inside the problem (`AddToProblem`).

```

% Unit identification
model.EI(...).Type          % Type of entity
model.EI(...).TagName       % Entity identifier
model.EI(...).DisplayName   % Short name of the entity. Used for display purposes.
model.EI(...).Description   % Long description of the entity
model.EI(...).Parent        % [main|any group or unit TagName] This field can be used to
    specify the TagName of the parent group or unit wich the unit belongs to.
model.EI(...).AddToProblem  % Defines if the entity shall be added to the problem.

```

Listing A.2: Generic fields for container entities


Layer	Description	Supported streams	Default
HeatCascade	Heat cascade	qt , vqt , ht , steam , vsteam	DefaultHeatCascade
Costing	Costs	cost	DefaultOpCost , DefaultInvCost , DefaultMechPower
MassBalance	Mass balances	mass	none
MassBalanceWithQuality	Mass balances, where stream mixes must satisfy a given quality	mq	none

Table A.2: Available layer types

Field	Var. type	Unit	Description
Type	cell string *		Defines the type of entity. For available types see this chapter content.
TagName	cell string		Entity identifier.
DisplayName	cell string		Short name of the entity. Used for display purposes.
Description	cell string		Long description of the entity.
Parent	cell string		[main any group or unit TagName] This field can be used to specify the TagName of the parent group or unit which the unit belongs to. When an element has a parent in another model, it is possible to set this field with the @ sign. When <i>OSMOSE</i> meets it, it reminds you to connect the element with a parent existing in the defined problem.
AddToProblem	bool	-	Defines if the entity shall be added to the problem. 0 allows to deactivate the entity, while 1 enables it.

Table A.1: Generic fields for container entities from listing A.2

A.3 Layers

Supported by: 

Layers are defined in Eiamp1 to represent streams behaviors. Each layer is a Mixed Integer Linear programming (MILP) model, composed by sets, parameters, variables and constraints. In Eiamp1, each stream is attached to a layer, depending on its type. Thermal streams are attached to heat cascade layers, while mass streams are attached to mass balance layers. The organization in layers allows to simultaneously solve several MILP problems, like combined heat and power production or handling streams which are at the same time mass and energy, like wastes.

Table A.2 reports the available layers types and the streams that it is able to handle. The name of the default Layer for each type is also given, if the energy integration software is able to automatically define one. If the user doesn't indicate a layer for a stream, the default one will be assigned by *OSMOSE* , by selecting the most appropriate default layer.

Listing A.3 shows the matlab syntax used to define a layer using the example of two layers. Fields defined in listing A.2 and table A.1 are used to define layers. Note that **Parent** is not used here, since layers are not hierarchically organized.

```

i=0;

i=i+1;
model.EI.Layers(i).Type      = {'HeatCascade'};      % This is for my regular qt
                        streams
model.EI.Layers(i).TagName   = {'DefaultHeatCascade'} % Loading the default HeatCascade
                        layer
model.EI.Layers(i).DisplayName = {'Default heat cascade for regular qt streams'}
model.EI.Layers(i).AddToProblem = 1; % I may want to deactivate this layer later...

i=i+1;
model.EI.Layers(i).Type      = {'MassBalance'}; % Several units use my fuel, so I want
                        to compute its distribution
model.EI.Layers(i).TagName   = {'FuelBalance'};
model.EI.Layers(i).DisplayName = {'This layer computes the mass balance of my fuel
                        streams'}
model.EI.Layers(i).AddToProblem = 1; % I may want to deactivate this layer later...

```

Listing A.3: `Layers` entities

A.4 Units

Several types of units are available: generic process or utility units and special units. Special units represent the most common equipment that can be found in processes, like Organic Rankine Cycles or Heat Pumps. Some other special units allow to define costs, operating conditions and other parameters of easy.

Units can be divided into three categories, resumed in table [A.3](#).















Category	Unit	Description
Generic	SYNPRO	  Defines a process units, where size and working conditions (temperatures and heat load) are fixed.
	SYNCYC2	  Defines a utility unit. Operating conditions are fixed, but the optimal size is computed by the energy integration software to satisfy the process heat requirements.
	SYNUT2	  Like for SYNCYC2 , the unit is optimally sized by the energy integration software. Moreover, this unit can contain only one stream. Its output temperature is also optimally computed.
Technologies	SYNRNK	 Defines a rankine cycle.
	SYNFRG	 Defines a refrigeration cycle
	SYNHEA	 Defines a steam network, including live steam, steam turbines, pumps, steam extraction and condensation.
	SYNFUEL	 Defines a fuel, which will be available for generating heat from combustion
Parameters	SYNUNT	 Defines parameters for the steam network
	EASYPRFS	 Defines parameter for the energy integration software Easy
	SYNINV	 Defines parameters for the calculation of heat exchangers area and cost
	SYNOPC	 Defins parameter for the computation of operating costs

Table A.3: Overview of available energy integration units

Like for streams, `EI.Units` is a structure that can contain several units. They are accessible by calling `EI.Units(1)`, `EI.Units(2)`, `EI.Units(3)`, and so on.

A.4.1 Generic fields for energy conversion units

Listing A.4 and table A.4 report fields that are commonly used in all energy conversion units. Some of them allow to identify the unit with a `Type` or a `TagName`, other define some economic or performance data, like `Cost1` or `Fmin`, or are related to the environmental performance, like `Impact2`, `UProcess` or `LCSType`. Fields defined in section A.2.2 are also available.

```

% Technical characteristics
model.EI.Units.ITY          % [0|1|2] Option used to choose the kind of flowrate to be
    computed
model.EI.Units.Fmin         % Minimum utilization rate of the unit [-] (if not specified
    , Fmin=1)
model.EI.Units.Fmax         % Maximum utilization rate of the unit [-] (if not specified
    , Fmax=1)
model.EI.Units.Mult         % Utilization rate of the unit.
model.EI.Units.Power2       % Mechanical power consumed by the utility
model.EI.Units.Power1       % Fixed mechanical power consumed when the utility is used.
    total mechanical Power = Power1 * Mult + Power2
model.EI.Units.Cost1        % Constant cost coefficient of the cycle
model.EI.Units.Cost2        % Proportional cost coefficient of the cycle Cost = Cost1 +
    Mult * Cost2
model.EI.Units.Cinv1        % Fixed investment cost coefficient
model.EI.Units.Cinv2        % Proportional investment cost coefficient
model.EI.Units.Impact1      % Constant environmental impact of the unit (if uprocess
    field not used)
model.EI.Units.Impact2      % Proportional environmental impact of the unit (if
    uprocess field not used)
model.EI.Units.UProcess     % Ecoinvent ID of the unit for impact calculation (if
    impact field not used)
model.EI.Units.LCSType      % Used along with 'UProcess' field, specifies if the impact
    is incremental ('s') or cumulated ('r')
model.EI.Units.LCSQuantity  % Quantity by which the impact calculated either with
    'Impact' or 'UProcess' should be multiplied
model.EI.Units.Location     % Defines the location where the unit is situated

```

Listing A.4: Process, utility and synut2 units

Field	Var. type	Unit	Description
Type	cell string *		SYNPRO , process , SYNCYC2 , utility , SYNUT2
ITY	int *	-	0 1 2 Option used to choose the kind of flowrate to be computed. 0 - if the flowrate and the integer variable has to be computed. 1 - if the integer variable is fixed, the flowrate is computed. 2 - if the flowrate and the integer variable are fixed. In this case, you can use Mult to set a unit size different from the nominal size.
Fmin	float	-	Minimum utilization rate of the unit
Fmax	float	-	Maximum utilization rate of the unit
Mult	float	-	Utilization rate of the unit. This value is computed if ITY is equal to 0 or 1 , or imposed if ITY is equal to 2 . This field represent the multiplication factor of the size of the unit with respect to the size you have declared by adding streams to the unit.
Power1	float	kW	Fixed mechanical power consumed when the utility is used. total mechanical power = Power1·Mult + Power2
Power2	float	kW	Mechanical power consumed by the utility
Cost1	float	\$	Constant cost coefficient of the cycle
Cost2	float	\$	Proportional cost coefficient of the cycle Cost = Cost1+Mult·Cost2
Cinv1	float	\$	Fixed investment cost coefficient
Cinv2	float	\$	proportional investment cost coefficient
Impact1	float	\$	Constant environmental impact
Impact2	float	\$	Proportional environmental impact
UProcess	cell string	\$	Ecoinvent ID of the unit for impact calculation
LCSType	cell string	\$	Specifies if the impact is incremental ('s') or cumulated ('r')
LCSQuantity	float	\$	Quantity to multiply nominal impact (calculated from ecoinvent dataset)
Location	cell string	\$	Defines the location where the unit is situated (sec. A.10)

A.4.2 Process units and utilities - `process|SYNPRO`, `utility|SYNCYC2` and `SYNUT2`

Supported by:  and 

`process`, `utilities` and `SYNUT2` units are defined with the same common syntax. `SYNUT2` units have some additional fields.

The difference between process and utilities is in their behavior. Process units are taken as is by the energy integration software, while utilities are resized to match the process requirements in terms of heat. The `ITY` field allows to change the way the utilities are sized. As explained in table A.4, a value of 2 allows to convert a utility into a process unit.

`SYNUT2` is a special utility, which can only contain a single stream. The unit is sized in the same way as a utility, in order to satisfy the heat requirements of the process units. Moreover, the output temperature of its stream can also be optimally computed inside a predefined range. This defines an additional degree of freedom, that the energy integration handles by discretizing the possible output temperatures into a finite number of possibilities. Fields `FirstT`, `LastT` and `Prec` allow to configure this additional degree of freedom.

Eiamp1 supports this unit type, which is transformed into a `SYNCYC2` unit. The stream contained in the unit is treated as a `vqt` (see sec. A.6.4) if it is a `qt` stream or as a `vsteam` (see sec. A.6.8) if it is a `steam` stream. `SYNUT2` units are thus deprecated, since `vqt` and `vsteam` can be inserted into `SYNPRO` and `SYNCYC2` units.

Listing A.4 and table A.4 report fields for `SYNPRO`, `SYNCYC2` and `SYNUT2` units. Moreover, listing A.5 and table A.5 shows the additional fields required to define `SYNUT2` units.

```
% fields used only for synut2
model.EI.Units.FirstT % Minimum output temperature
model.EI.Units.LastT  % Maximum output temperature
model.EI.Units.Prec    % Discrete temperature interval
```

Listing A.5: `SYNUT2` units

Field	Var. type	Unit	Description
<code>FirstT</code>	<code>float</code>	K	Minimum output temperature
<code>LastT</code>	<code>float</code>	K	Maximum output temperature
<code>Prec</code>	<code>float</code>	K	Discrete temperature interval

Table A.5: `SYNUT2` fields from listing A.5

A.4.3 Rankine cycles - `SYNRNK`

Supported by: 

`SYNRNK` units define Rankine cycles. The cycle are computed using the thermodynamic model of the refrigerants embedded in easy. The thermodynamic model used is Lee-Kesler with the thermodynamic data properties from BELSIM-VALI.

Listing A.6 and table A.6 define the syntax of the unit. Fields from listing A.4 and table A.4, with exception for `Power1` and `Power2`, are also defined for this unit.

```

model.EI.Units.EvapTemp           % Temperature of the evaporation
model.EI.Units.EvapSuperhTemp     % Temperature of the superheating
model.EI.Units.CondTemp           % Temperature of the condensation
model.EI.Units.DTmin_2            % Temperature difference contribution
model.EI.Units.DTevap             % DTmin/2 for the evaporation
model.EI.Units.DTcond             % DTmin/2 for the condenser
model.EI.Units.Refrigerant        % Name of the refrigerant to be selected in the list of
    refrigerant
model.EI.Units.ReferenceFlow      % Reference refrigerant mass flow
model.EI.Units.TurbPower          % Turbine power
model.EI.Units.TurbMechEff        % Turbine Mechanical efficiency
model.EI.Units.TurbIsEff          % Turbine isentropic efficiency
model.EI.Units.PumpPower          % Pump power
model.EI.Units.PumpMechEff        % Pump mechanical efficiency
model.EI.Units.PumpIsEff          % Pump isentropic efficiency

```

Listing A.6: Rankine cycle (SYNRNK) units

Field	Var. type	Unit	Description
Type	cell string *		SYNRNK
EvapTemp	float	K	Evaporation temperature
EvapSuperhTemp	float	K	Superheating temperature
CondTemp	float	K	Condensation temperature
DTmin_2	float	ΔK	Temperature difference contribution
DTevap	float	ΔK	DTmin/2 for the evaporator
DTcond	float	ΔK	DTmin/2 for the condenser
Refrigerant	cell string *		Name of the refrigerant to be selected in the list from table A.7
ReferenceFlow	float	kg/s	Reference refrigerant mass flow
TurbPower	float	kW	Turbine power
TurbMechEff	float	-	Turbine Mechanical efficiency
TurbIsEff	float	-	Turbine isentropic efficiency
PumpPower	float	kW	Pump power
PumpMechEff	float	-	Pump mechanical efficiency
PumpIsEff	float	-	Pump isentropic efficiency

Table A.6: SYNRNK fields from listing A.6

The list of available refrigerant is given on table A.7.

Refrigerants	EAU, R702, R720, R728, R740, R732, R50, R784, R704, R14, R1150, R170, R744, R23, R13, R116, R1270, R13B1, R290, R143, R717, R12, R22, R115, R218, R152, R40, R600A, R764, R630, R142, R600, R12B1, R160, R114, R21, STEAM, R245FA, R245CA, RC318, PP50, PP5, R32, R134A, R125, R407C, R123
--------------	--

Table A.7: List of refrigerants

A.4.4 Refrigeration and heat pump cycles - SYNFRG

Supported by: 

SYNFRG units define a refrigeration cycle or a heat pump. The cycle is computed using the thermodynamic model of the refrigerants (see table A.7) embedded in easy. The thermodynamic model is

Lee-Kesler with the thermodynamic data properties from BELSIM-VALI. The parameters for the definition of the unit are given on table A.7. Fields defined in listing A.4 and table A.4 are also available, except for `Power1` and `Power2`.

```
%% Fields for SYNFRG units

% General
model.EI.Units.DTmin_2      % Temperature difference contribution
model.EI.Units.Refrigerant  % Name of the refrigerant to be selected in the list of
    refrigerant
model.EI.Units.ReferenceFlow % Reference refrigerant flow
model.EI.Units.ReferencePower % Reference compressor power

% Evaporator
model.EI.Units.EvapTemp      % Evaporation temperature
model.EI.Units.EvapTemp2     % This temperature can be defined as upper bound to create
    several cycles having an evaporation temperature defined in the interval EvapTemp:
    EvapTempInt:EvapTemp2
model.EI.Units.EvapTempInt   % Temperature interval allowing to define the number of
    cycles to create.
model.EI.Units.EvapDT        % DTmin/2 for the evaporator
model.EI.Units.EvapSuperhDT  % Temperature difference of the superheating at the exit
    of the evaporation

model.EI.Units.EvapLoad      % Evaporator Load
model.EI.Units.EvapPress     % Evaporator pressure
model.EI.Units.EvapHTC       % Heat transfer coefficient of the evaporation
model.EI.Units.EvapArea      % Evaporator Area
model.EI.Units.EvapCost      % Evaporator cost
model.EI.Units.EvapFlow      % Reference flow in the evaporator

% Condenser
model.EI.Units.CondTemp      % Condensation temperature
model.EI.Units.CondTemp2     % This temperature can be defined as upper bound to create
    several cycles having a condensation temperature defined in the interval CondTemp:
    CondTempInt:CondTemp2
model.EI.Units.CondTempInt   % Defines the discrete temperature interval to be used to
    create several condensation temperatures.
model.EI.Units.CondDT        % DTmin/2 for the condenser
model.EI.Units.CondUndercDT  % Temperature difference of the undercooling
model.EI.Units.CondHeatLoad  % Condenser heat load
model.EI.Units.CondPress     % Condenser pressure
model.EI.Units.CondHTC       % Heat transfer coefficient of the condenser
model.EI.Units.CondArea      % condenser area
model.EI.Units.CondCost      % Condenser cost
model.EI.Units.CondFlow      % Reference condenser flow

% Compressor
model.EI.Units.ComprPower     % Compressor power
model.EI.Units.ComprStages    % Numer of stages of the compressor
model.EI.Units.ComprIsEff     % Compressor isentropic efficiency
model.EI.Units.ComprCost      % Compressor cost
```

Listing A.7: Refrigeration cycle definition

Note that when two temperatures are given several cycles are created automatically by easy.

Field	Var. type	Unit	Description
Type	cell string *		SYNFRG
DTmin_2	float	ΔK	Temperature difference contribution
Refrigerant	cell string *		Name of the refrigerant to be selected in the list given in table A.7
ReferenceFlow	float	kmol/s	Reference refrigerant flow. This field allows to set the nominal size of the cycle. If the refrigerant flow is not known, it is possible to set the reference size with the compressor power ReferencePower .
ReferencePower	float	kW	Reference compressor power. This field allows to set the nominal size of the cycle. If the compressor power is not known, it is possible to set the nominal size with the refrigerant flow mltabReferenceFlow.
EvapTemp	float	K	Evaporation temperature
EvapTemp2	float	K	This temperature can be defined as upper bound to create several cycles having an evaporation temperature defined in the interval EvapTemp:EvapTempInt:EvapTemp2. For example, 300:320:10 create cycle with evaporation temperature of successively 300, 310, 320 K
EvapTempInt	float	ΔK	Defines the discrete temperature interval to be used to create several evaporation temperatures.
EvapDT	float	ΔK	DTmin/2 for the evaporator
EvapSuperhDT	float	ΔK	Temperature difference of the superheating at the exit of the evaporation
EvapLoad	float	kW	Evaporator Load
EvapPress	float	bar	Evaporator pressure
EvapHTC	float	$kW/m^2 K$	Heat transfer coefficient of the evaporation
EvapArea	float	m^2	Evaporator Area
EvapCost	float	\$	Evaporator cost
EvapFlow	float	kmol/s	Reference flow in the evaporator
CondTemp	float	K	Condensation temperature
CondTemp2	float	K	This temperature can be defined as upper bound to create several cycles having a condensation temperature defined in the interval CondTemp:CondTempInt:CondTemp2. For example, 300:320:10 create cycle with condensation temperature of successively 300, 310, 320 K
CondTempInt	float	ΔK	Defines the discrete temperature interval to be used to create several condensation temperatures.
CondDT	float	δK	DTmin/2 for the condenser
CondUndercDT	float	ΔK	Temperature difference of the undercooling
CondHeatLoad	float	kW	Condenser heat load
CondPress	float	bar	Condenser pressure
CondHTC	float	$kW/m^2 K$	Heat transfer coefficient of the condensation
CondArea	float	m^2	condenser area
CondCost	float	\$	Condenser cost
CondFlow	float	kmol/s	Reference condenser flow
ComprPower	float	kW	Compressor power
ComprStages	int	-	Number of stages of the compressor
ComprIsEff	float	-	Compressor isentropic efficiency

Continued on next page ...

... Continuing from previous page

Field	Var. type	Unit	Description
ComprCost	float	\$	Compressor cost

Table A.8: **SYNFRG** fields from listing **A.7**

A.4.5 Steam network units - **SYNHEAx**

Supported by: 

There are two different ways of defining steam networks:

1. Definition of a normal group and definition of **SYNHEAx** unit for each header. This way is presented in this section.
2. Definition of a complete steam network (including all required elements) as a type of group (see also section **A.5.1**).

A generic steam network may be composed of several elements:

- Steam headers producing live steam
- Steam headers distributing steam
- A condensation level, returning to the steam generator
- Pumps, to pressurize water
- Turbines to expand steam and produce electricity

EnergyTechnologies provides a full syntax to define steam networks. The energy integration software is able to optimally size the steam network by computing the flowrate (and the size) of each header, pump and turbine in order to satisfy the process requirements and to maximise the electricity production.

A minimal steam network must respect following conditions:

1. At least one steam production level **SYNHEAV**
2. At least one steam consumption level **SYNHEAU**
3. At least one condensation level **SYNHEAC**
4. The minimum pressure level observed in the list of headers must be of condensation type **SYNHEAC**

In eiampl it is possible to define a **ElectricityBalanceLayer** field to define the name of the **MassBalance** layer where an electricity balance is performed. This is useful when interactions on a global electricity market is computed with **mass** streams.

Listing **A.8** and table **A.16** detail the basic syntax of **SYNHEA** units. Listing **A.9** and table **A.10** detail some additional field of this unit. Fields defined in section **A.2.2** are also available.

```

model.EI.Units.Type          % 'SYNHEAC' for liquid state header
                             % 'SYNHEAV' for steam production headers
                             % 'SYNHEAU' for steam consuming headers
                             % 'SYNHEA'  for normal header with steam consumption and
                             usage

% Header properties
model.EI.Units.Pressure      % Pressure of the header
model.EI.Units.Temperature   % temperature of the header
model.EI.Units.SuperheatingDT % superheating temperature difference
model.EI.Units.Vapf          % Vapor fraction of the header (1 for vapor stream, 0 for
                             liquid)
model.EI.Units.DTmin_2       % DTmin/2 contribution of the steam
model.EI.Units.ElectricityBalanceLayer % (optional) For Eiampl only. Defines the name of
                             the MassBalance layer where a electricity balance is performed.

% Optional eiampl fields
model.EI.Units.HeaderName    % To define the name of the generated layer, in
                             order to connect other streams than those in the synhea family

```

Listing A.8: SYNHEAx header definition


Field	Var. type	Unit	Description
Type	cell string *		Defines the type of unit. SYNHEAC defines a header in liquid state. It represents the condensate return, just before the pump. SYNHEAV defines a steam production header. It provides steam to the network. SYNHEAU defines a steam consumption header. It recovers steam from the network and fills the return condensate, unless steam is used elsewhere (e.g. in the process).
Pressure	float	bar	Header pressure.
Temperature	float	K	Header temperature. If SuperheatingDT is set, this field is not necessary.
SuperheatingDT	float	ΔK	Superheating temperature. If Temperature is set, this field is not necessary.
Vapf	float	-	Vapour fraction. This field is necessary only when steam is under the saturation curve.
DTmin_2	float	ΔK	DTmin/2 contribution of the stream.
ElectricityBalanceLayer	cell string	 (optional)	Defines the name of the MassBalance layer where a electricity balance is performed.

Table A.9: Steam headers SYNEHAX syntax from listing A.8

```

model.EI.Units.C1ALPRO % feed import fixed cost
model.EI.Units.C2ALPRO % feed import proportional cost
model.EI.Units.MINALPRO % feed import min flow
model.EI.Units.MAXALPRO % feed import max flow
model.EI.Units.FALPROC % feed import flow (result)
model.EI.Units.TYPALPRO % feed import type {0|1|2}

model.EI.Units.C1ALEX % existing feed to the header other definitions as above
model.EI.Units.C2ALEX % existing feed to the header
model.EI.Units.MINALEX % existing feed to the header
model.EI.Units.MAXALEX % existing feed to the header
model.EI.Units.FALEX % existing feed to the header
model.EI.Units.TYPALEX % existing feed to the header

model.EI.Units.C1ALNEW% new feed to the header
model.EI.Units.C2ALNEW% new feed to the header
model.EI.Units.MINALNEW% new feed to the header
model.EI.Units.MAXALNEW% new feed to the header
model.EI.Units.FALNEW % new feed to the header
model.EI.Units.TYPALNEW % new feed to the header

model.EI.Units.C1EXCND % existing condensation
model.EI.Units.C2EXCND % existing condensation
model.EI.Units.MINEXCND % existing condensation
model.EI.Units.MAXEXCND % existing condensation
model.EI.Units.FEXCND % existing condensation
model.EI.Units.TYPEXCND % new condensation

model.EI.Units.C1NEWCND % new condensation
model.EI.Units.C2NEWCND % new condensation
model.EI.Units.MINEWCND % new condensation
model.EI.Units.MAXNEWCN % new condensation
model.EI.Units.FNEWCND % new condensation
model.EI.Units.TYPNEWCN % new condensation

model.EI.Units.C1DOFF % draw off fixed cost
model.EI.Units.C2DOFF % draw off proportional cost
model.EI.Units.MINDOFF % draw off min flow
model.EI.Units.MAXDOFF % draw off max flow
model.EI.Units.DRAWOFF % draw off flow
model.EI.Units.TYPDOff % draw off type (0|1|2)


```

Listing A.9: Advanced SYNHEAx header definition

Field	Var. type	Unit	Description
Type	cell string *		Defines the type of unit. SYNHEAC defines a header in liquid state. It represents the condensate return, just before the pump. SYNHEAV defines a steam production header. It provides steam to the network. SYNHEAU defines a steam consumption header. It recovers steam from the network and fills the return condensate, unless steam is used elsewhere (e.g. in the process). SYNHEA defines a generic header. TODO mieux expliquer
TODO	EnergyTechnologies		TODO

Table A.10: Steam headers SYNEHAX advanced syntax from listing A.9

A.4.6 Fuels - SYNFUEL



Supported by: 

This unit is defined as a container for all the fuels defined in the problem. Details about the definition of fuels is given at section A.6.6.

The syntax of this unit is completely defined by the fields defined in section A.2.2. The unit type is `model.EI.Units.Type = 'SYNFUEL'`.

TODO: expliquer comment on utilise un combustible pour generer une combustion dans une chaudiere ou dans un autre utilitaire.

A.4.7 Steam network parameters - SYNUNT

Supported by:  and 

This unit defines parameter for the steam network (see sec. A.4.5 and A.5.1), including efficiencies of the steam turbines. Listing A.10 and table A.11 provides the syntax to be used. One unit is enough for all steam networks. The parameters are global, i.e. they are defined with the same value for all the networks. Fields defined in section A.2.2 are also available.

```
model.EI.Units.Type = {'SYNUNT'}           % Type of unit
model.EI.Units.HeadersMaxNumber           % Maximum number of headers
model.EI.Units.TurbExpRatioMin             % Minimum expansion ratio of the turbines
model.EI.Units.TurbPowerMin               % Minimum power for creating an expansion turbine (
    recommended to be set to 0)
model.EI.Units.TurbPowerMax               % Maximum power for an expansion
model.EI.Units.TurbBackPressEff           % Isentropic efficiency of the expansion for back
    pressure turbines
model.EI.Units.TurbCondEff                % Isentropic efficiency of the condensing turbines
model.EI.Units.PumpEff                    % EIAMPL only. Isentropic efficiency of water pumps
model.EI.Units.ElecToMechPower            % Defines whether electricity must be converted into
    mechanical power, if required by the process
model.EI.Units.MechPowerToElec            % Defines whether the mechanical power must be
    converted into electricity
model.EI.Units.Retrofit                   % Indicated if retrofit has to be calculated for steam
    turbines.
```

Listing A.10: SYNUNT Units

A.4.8 Easy parameters - EASYPRFS

Supported by: 

This unit holds the parameters used by Easy, the energy integration software, to run computations. Listing A.11 and table A.12 detail its syntax. Other parameters are available in Easy, but they are not detailed here, because they are related to functionalities unsupported by *OSMOSE*.

Field	Var. type	Unit	Description
Type	cell string *		EASYPRFS . Defines the type of unit.
RadiationTemp	float	K	Temperature used for the separating radiative and convective section in combustion

Continued on next page ...

... Continuing from previous page

Field	Var. type	Unit	Description
AmbientTemp	float	K	Ambient temperature. This value is used to separate cooling requirements in two zones: ambient cooling (with air or water, for example) and refrigeration cooling.
ExergyRefTemp	float	K	Temperature used as a reference for the exergy (default = 298K)
ObjFunction	cell string *		Defines the objective function to be computed. This value is overridden by <i>OSMOSE</i> variable o.Easy.Objective and is thus deprecated. Table A.13 lists all possible values.
NumberOfSolutions	int		Maximum number of different solutions to find.
Unitref	int		Defines the load scenario to be considered if Unitref contains 0 all the load scenarios will be considered and a computation of multi load scenario optimisation will be performed.
GridElecEff	float	-	Defines the efficiency of the grid production (default=0.55)
GridCO2	float	kg/kJ	Defines the amount of CO2 released for the production of electricity from the grid.

Table A.12: Easy parameters **EASYPRFS** fields from listing A.11

Value	Description
1	Minimize operating cost
2	Minimize Exergy losses
3	Minimize investment cost
4	Maximize net present value
5	Maximise mechanical power production
6	Minimize Total Energy
7	Minimize Total CO2 emissions
8	Minimize conversion exergy losses

Table A.13: Objective functions for Easy defined via the **ObjFunction** field

A.4.9 Costing unit - **SYNCOST**

Supported by: **Easy**

TODO

A.4.10 Investment cost parameters - **SYNINV**

Supported by: **Easy**

The energy integration software provides an estimation of the heat exchanger area and cost, when an energy integration is computed. The cost of the heat exchangers is computed with a generic exponential expression, as shown in equation A.1.

$$C_{ex} = a_{ex} + b_{ex} \cdot (A)^{c_{ex}} \quad (\text{A.1})$$

Cost C_{ex} depends on three constants a_{ex} , b_{ex} , c_{ex} and from the exchangers area A . These constants can be configured using the **SYNINV** unit, which is detailed in listing A.12 and table A.14. Fields defined


Field	Var. type	Unit	Description
Type	cell string *		SYNUNT . Defines the type of unit.
HeadersMaxNumber	int	-	Maximum number of production headers. If more than this number is defined in the steam network, only the best HeadersMaxNumber will be retained.
TurbExpRatioMin	float	-	Minimum expansion ratio of turbines. If a turbine cannot meet this ration, it won't be retained in the solution.
TurbPowerMin	float	kW	Minimum power of steam turbines
TurbPowerMax	float	kW	Maximum power for an expansion
TurbBackPressEff	float	-	Isentropic efficiency of the expansion for back pressure turbines
TurbCondEff	float	-	Isentropic efficiency of the condensing turbines
PumpEff	float	-	 Isentropic efficiency of water pumps
ElecToMechPower	int *	-	Defines whether electricity must be converted into mechanical power, if required by the process. 1 if electricity might be imported into the network for a given mechanical power production (used if MechPowerToElec=1). 0 if either electricity or the Network should be used to produce a given mechanical power.
MechPowerToElec	int *	-	Defines how mechanical power produced by turbines must be used. A value of 0 indicates electricity production, 1 indicates mechanical power for other units requiring it.
Retrofit	int *	-	1 if retrofit has to be calculated: When new turbines have to be used, their cost will be calculated using the SYNCOST unit parameters (see sec. A.4.9).

Table A.11: SYNUNT fields from listing A.10

```

model.EI.Units.Type = {'EASYPRFS'}           % Type of unit
model.EI.Units.RadiationTemp
model.EI.Units.AmbientTemp
model.EI.Units.ExergyRefTemp
model.EI.Units.ObjFunction
model.EI.Units.NumberOfSolutions
model.EI.Units.Unitref
model.EI.Units.GridElecEff
model.EI.Units.GridCO2

```

Listing A.11: Easy parameters - EASYPRFS Unit

in section A.2.2 are also available.

The area of a heat exchanger is computed with the relation defined by equation A.2. Q is the heat load, ΔT_{lm} is the log-mean temperature difference and U is the overall heat exchange transfer coefficient, which is function of the heat transfer coefficient of all the streams crossing the exchanger h_i , as shown by relation A.3.

$$A = \frac{Q}{U \Delta T_{lm}} \quad (\text{A.2})$$

$$\frac{1}{U} = \sum_i \frac{1}{h_i} \quad (\text{A.3})$$

So, the energy integration software need the heat transfer coefficient of each stream in order to compute the heat transfer area. But this value is not always provided by the user, since this field is sometimes optional (see for example, `qt` streams at section A.6.3). When this value is not available, relation A.4 is used to estimated it.

$\Delta T_{min/2,ref}$ and h_{ref} are two reference values, that can be set by the user using respectively `Dtref` and `Meanhtf` in the `SYNINV` unit.

$$\frac{\Delta T_{min/2,i}}{\Delta T_{min/2,ref}} = \left(\frac{h_i}{h_{ref}} \right)^{c_{ex}} \quad (\text{A.4})$$

```
% fields used to define SYNINV units
model.EI.Units.Meanhtf      % Meanhtf is the reference heat transfer film coefficient
                             associated to the reference DTmin/2 value
model.EI.Units.Dtref        % Reference DTmin/2
model.EI.Units.Aex          % Aex, Bex and Cex are used to estimate the cost of the
                             heat exchanger with a formula like Cost = Aex +Bex(Area)^Cex
model.EI.Units.Bex
model.EI.Units.Cex
```

Listing A.12: SinInv Units

Field	Var. type	Unit	Description
<code>Meanhtf</code>	<code>float</code>	kW/K m ²	Meanhtf is the reference heat transfer film coefficient associated to the reference <code>Dtref</code> value.
<code>Dtref</code>	<code>float</code>	ΔK	Reference DTmin/2.
<code>Aex,Bex,Cex</code>	<code>float</code>	-	Constants used to estimate the cost of heat exchangers.

Table A.14: Heat exchanger parameters - `SYNINV` A.12

A.4.11 Operating cost parameters - `SYNOPC`

Supported by: 

This unit allows to define parameters for the computation of basic operating cost of utilities. Listing A.13 and table A.15 report the syntax. Fields from section A.2.2 are also available.

Note that `HotUtility`, `ColdUtility`, `RefrigerationUtility` allow to define operating cost for so-called default utilities. The usage of defaults values for parameters is deprecated, since they are subject to change and your model may become unstable. To program a good model, all the parameters should be set by the user. In the case of utilities (for example of `SYNCRYC2` type), you should prefer to directly set costing parameters using `Cost1` and `Cost2` fields.

```

% fields used to define SYNOPC units

model.EI.Units.Type           % Type of unit
model.EI.Units.OpTime         % Total operating time of the installation (in hours)
model.EI.Units.Ref            % Reference operating cost of the unit in the given period
model.EI.Units.ElecIn         % Price of electricity purchase (CHF/GJ)
model.EI.Units.ElecOut        % Selling price of the electricity (CHF/GJ)
model.EI.Units.ElecMaxIn      % Maximum electrical power at import
model.EI.Units.ElecMaxOut     % Maximum electrical power at export
model.EI.Units.HotUtility     % Cost of the default hot utility
model.EI.Units.ColdUtility    % Cost of the default cold utility
model.EI.Units.RefrigerationUtility % Cost of the default refrigeration utility

```

Listing A.13: Operating cost parameters

Field	Var. type	Unit	Description
Type	cell string *		SYNOPC .
OpTime	float	h	Total operating time.
Ref	float	\$/GJ	Reference operating cost. TODO: a quoi ca sert?
ElecIn	float	\$/GJ	Price of electricity purchase.
ElecOut	float	\$/GJ	Selling price of the electricity.
ElecMaxIn	float	kW	Maximum electrical power at import.
ElecMaxOut	float	kW	Maximum electrical power at export
HotUtility	float	\$/GJ	Cost of the default hot utility
ColdUtility	float	\$/GJ	Cost of the default cold utility
RefrigerationUtility	float	\$/GJ	Cost of the default refrigeration utility

Table A.15: Operating cost parameter fields from listing A.13

A.5 Groups

Supported by: 

It is sometimes useful to group several units and equations into one single entity. The `EI.Groups` field allows to group `EI.Units`, `EI.Equations` and even `EI.Groups`¹.

Grouping entities inside `Groups` offers two major advantages:

1. All the streams of a group are plotted together in an integrated composite curve. So, for example, if you have cold water and steam grouped in a “utility” group, you can plot their integrated composite curves to show their integration with respect to the process streams.
2. When computing an energy integration, there may be several solutions leading to the same optimal objective function value. To select the best one, the energy integration software promotes heat exchange firstly inside streams of the same unit, then for streams inside the same group and finally for all the streams of the whole problem. So, to encourage heat exchange between certain units, it is a good idea to group them inside a `Group`.

The syntax of `Goups` is very simple and correspond to the fields defined in section A.2.2. An example is `model.EI.Groups.TagName`.

¹`EI.Groups` can be included into other `EI.Groups`

A.5.1 Steam Networks - `SteamNetwork`

See also section [A.4.5](#).

Steam networks can also be defined as a group type. This type of groups generate steam networks with different levels. The syntax for the group definition is given in [A.14](#). This way of describing steam networks has the advantage that EnergyTechnologies is able to check the syntax and identify eventually wrong definition of this type of group.

```

ng = ng + 1;
technology.EI.Groups(ng).Type = {'SteamNetwork'} ; %Type of group
technology.EI.Groups(ng).TagName = {'steam_network'};
technology.EI.Groups(ng).DisplayName = {'steam network'} ;
technology.EI.Groups(ng).Parent = {'utility'};
technology.EI.Groups(ng).AddToProblem = {'1'};

nl = 0;
nl = nl + 1;
technology.EI.Groups(ng).Levels(nl).Type = {''}; % 'production' for steam production headers
                                                % 'usage' for steam consuming headers
                                                % 'condensation' for liquid state header
                                % 'both' for consuming et production header (easy convergence is not
                                guaranteed)
                                % 'userdefined' for header specification defined by user

technology.EI.Groups(ng).Levels(nl).Pressure % Pressure of the header
technology.EI.Groups(ng).Levels(nl).SuperheatingDT % superheating temperature of the
header
technology.EI.Groups(ng).Levels(nl).Vapf % Vapour fraction of the header (1 for vapour
stream, 0 for liquid stream) % SuperheatingDT or Vapf. The others are deduced by easy
technology.EI.Groups(ng).Levels(nl).DTmin_2 = % DTmin/2 contribution of the stream

% These fields are optional, they can be set to add a reheat
technology.EI.Groups(ng).Levels(nl).doReheat = 1 ;
technology.EI.Groups(ng).Levels(nl).ReheatTemperature = {'-1'} ; % Don't set this field if
ReheatSuperheatingDT is defined. Vapf is not available for reheated streams.
technology.EI.Groups(ng).Levels(nl).ReheatSuperheatingDT = 100 ;
technology.EI.Groups(ng).Levels(nl).ReheatDTmin_2 = 4 ;

```

Listing A.14: `SteamNetwork` group definition

Field	Var. type	Unit	Description
Type	cell string *		Defines the type of unit. condensation defines a header in liquid state. It represents the condensate return, just before the pump. production defines a steam production header. It provides steam to the network. usage defines a steam consumption header. It recovers steam from the network and fills the return condensate, unless steam is used elsewhere (e.g. in the process).
Pressure	float	bar	Header pressure.
SuperheatingDT	float	ΔK	Superheating temperature. If Temperature is set, this field is not necessary.
Vapf	float	-	Vapour fraction. This field is necessary only when steam is under the saturation curve.
DTmin_2	float	ΔK	DTmin/2 contribution of the stream.

Table A.16: Steam network levels syntax from listing A.8

A.6 Streams

Several types of streams are available in the **EI** definition. Some are generic definitions, other are based on a thermodynamic model of the most common substances used in energy conversion units.

Section A.2.2 defines the common syntax, available for all the types of streams. Listing A.15 provides an additional field, allowing to define a stream with a shorter syntax. Note that *OSMOSE* processes the definition and always returns an extended syntax. The short syntax can be used in replacement for the extended one. No mix of both syntaxes is allowed. If *OSMOSE* finds the short syntax, it won't check for available fields in the extended syntax.

Other fields will appear depending on the type of stream. The keyword **EI.Streams** is used to identify streams. Note that this structure can contain several streams, which will be available by recovering **EI.Streams(1)**, **EI.Streams(2)**, **EI.Streams(3)**, and so on.

```
% Abridged streams syntax
model.EI.Streams.Short = {} % Short definition of a stream. Its content depend
    on the stream type.
```

Listing A.15: Generic definition of a stream in **model.EI** syntax

A.6.1 Overview of available streams

Table A.17 resumes the available streams. Following sections will provide the exact syntax to be used to define them.















Category	Unit	Description
Generic	qt	  Defines a generic thermal stream, changing temperature linearly.
	vqt	 Similar to qt streams. Output temperature can be specified within a range. Optimal value is computed by EI software.
	ht	 Defines a generic thermal stream, expressed as a piecewise linear line.
	cost	 Represents a cost.
	mass	 Defines mass streams
	mq	 Defines mass streams with a quality (ex. concentration)
	mqpartial	 Defines mass streams with partial flowrates
Material	fuel	 Defines a fuel, to be used to generate heat within a combustion.
	steam	  Defines a stream composed by water and/or steam.
	vsteam	 Similar to steam streams. Output temperature can be specified within a range. Optimal value is computed by EI software.
	stream	  Defines a stream composed by a known substance (see tab. A.23).

Table A.17: Overview of available streams

A.6.2 Note about layers

Supported by: 

In Eiempl, each stream must belong to a Layer (see sec. A.3). If no layer is defined the Eiempl will define one by default, when possible.

An additional field **Layer** can be appended to the definition of each stream to indicate the name of the Layer it belongs. By default the EI software automatically detects the right Layer for each stream from its Type.

A.6.3 Thermal Streams - qt

Supported by:  and 

The structure shown in listing A.16 defines a generic thermal stream. This stream requires two temperatures, two enthalpies, a minimum temperature approach and, optionally, a heat transfer coefficient. We define stream inlet and outlet the points where the measurements are taken. The flow of the stream must run from inlet to outlet.

Table A.18 details each field. Generic fields defined in section A.6 are also available.

```

% Long definition
model.EI.Streams.Type = {'qt'}; % Type of stream.
model.EI.Streams.Hin   % Inlet flow enthalpy of the stream [kW]
model.EI.Streams.Hout  % Outlet flow enthalpy of the stream [kW]
model.EI.Streams.Tin   % Inlet temperature [K]
model.EI.Streams.Tout  % Outlet temperature [K]
model.EI.Streams.DTmin_2 % Minimum temperature approach [K]
model.EI.Streams.h     % (optional) heat transfer coefficient [kW/Km^2]

% Abridged
model.EI.Streams.Short = {'qt','Parent','TagName',Tin,Hin,Tout,Hout,DTmin_2,h}

% Example of abridged definition
model(1).EI.Streams(3).Short = {'qt','myUnit','myStream',1200,'443','0','myTag',3} % MyTag is a
Tag coming for example from another model

```

Listing A.16: QT stream

Field	Var. type	Unit	Description
Type	cell string *		qt
Hin	float	kW	Inlet enthalpy.
Hout	float	kW	Outlet enthalpy.
Tin	float	K	Inlet temperature.
Tout	float	K	Outlet temperature.
DTmin_2	float	ΔK	DTmin/2
h	float	kW/K m ²	Optional. Heat transfer coefficient.

Table A.18: qt fields from listing A.16

A.6.4 Variable Thermal Streams - vqt

These streams represent a thermodynamic transformation, where no phase change takes place. They are similar to qt streams and they differ in the definition of the output temperature, which is defined within a discrete range instead of a fixed point. The energy integration software will chose the best output temperature depending ot the integration problem to be solved.

vqt streams are defined exactly as qt streams (see listing A.16, table A.18 and section A.6) plus some additional fields, described in listing A.17 and table A.19.

Fields Tin, Tout, Hin and Hout are used as initial conditions to deduce stream $\dot{m}c_p$.

Field	Var. type	Unit	Description
FirstT	float	K	Lowest accepted output temperature.
LastT	float	K	Highest accepted temperature.
Prec	float	-	Discrete temperature interval between to possible output temperatures.

Table A.19: vqt fields from listing A.17

A.6.5 Piecewise thermal Streams - ht

Supported by: 

```
% Example of vqt stream
ns = ns+1;
model.EI.Streams(ns).Type = {'vqt'};
model.EI.Streams(ns).TagName = {'vqt_utility2'};
model.EI.Streams(ns).DisplayName = {'vqt stream'};
model.EI.Streams(ns).AddToProblem = 1;
model.EI.Streams(ns).Parent = {'hot'};
model.EI.Streams(ns).FirstT = {'@cu2_tin+(@cu2_tout-@cu2_tin)/5'};
model.EI.Streams(ns).LastT = {'@cu2_tout'};
model.EI.Streams(ns).Prec = {'(@cu2_tout-@cu2_tin)/20'};
model.EI.Streams(ns).Tin = {'@cu2_tin'};
model.EI.Streams(ns).Tout = {'@cu2_tout'};
model.EI.Streams(ns).Hin = 50;
model.EI.Streams(ns).Hout = 0;
model.EI.Streams(ns).DTmin_2 = 5;
```

Listing A.17: VQT stream

ht streams are more complex version of **qt** (sec. A.6.3) streams. They represent a stream changing temperature following a linear path. Stream c_p may vary during the transformation. This is expressed as a piecewise linear path.

The structure shown in listing A.18 shows an example definition of a **ht** stream. The syntax to be used is the same as the one defined for **qt** streams, except for the **Type** field, which is **ht**.


This stream requires at least two temperatures, two enthalpies, a minimum temperature approach. Output temperature, enthalpies and $\Delta T_{min}/2$ may be expressed as vectors having all the same length (except for $\Delta T_{min}/2$, which may be a scalar). Each value in these vectors is a starting/end point of a linear segment. If **Hin** is a scalar, **Hout** is considered as a series of incremental values. Otherwise, the enthalpy difference of each linear segment is calculated as the difference between outlet and inlet enthalpy.

We define stream inlet and outlet the points where the measurements are taken. The flow of the stream must run from inlet to outlet.

```
% Long definition
model.EI.Streams.Type = {'ht'};           % Type of stream.
model.EI.Streams.Hin = [0,0,0];           % Inlet flow enthalpy of the stream [kW]
model.EI.Streams.Hout = [20,100,130];     % Outlet flow enthalpy of the stream [kW]
model.EI.Streams.Tin = 293;               % Inlet temperature [K]
model.EI.Streams.Tout = [315,500,600];    % Outlet temperature [K]
model.EI.Streams.DTmin_2 = 5;             % Minimum temperature approach [K]
```

Listing A.18: HT stream

A.6.6 Fuels - fuel

Supported by: 

This stream allows to define a fuel, to be used as heat source generated by a combustion. Listing A.19 and table A.20 shows the syntax. Fields from section A.2.2 are also available. The **Parent** field must point to the tagname of the **SYNFUEL** (see sec. A.4.6) defined for the problem.

```

%% fields used to define SYNFOEL units
model.EI.Streams.Type           % Type of stream
model.EI.Streams.LHV            % Fuel Lower Heating Value
model.EI.Streams.AdiabaticTemp  % Combustion adiabatic temperature
model.EI.Streams.StackTemp      % Minimum accepted temperature at the stack
model.EI.Streams.State          % State of fuel (0 gas, 1 solid, 2 liquid)
model.EI.Streams.Pressure       % Fuel pressure (required for a gaseous fuel)
model.EI.Streams.Molw           % Fuel molar weight
model.EI.Streams.FlowMax        % Maximum fuel mass flow
model.EI.Streams.Cost           % Fuel cost

%Composition
model.EI.Streams.S              % Sulphur content of the fuel
model.EI.Streams.N              % Nitrogen content of the fuel
model.EI.Streams.O2             % Oxygen mass flow consumed for a complete combustion of 1
                                kg of fuel

% Fumes and dust
model.EI.Streams.CO2            % Mass of CO2 created for a complete combustion of 1 kg of
                                fuel
model.EI.Streams.H2O            % Amount of water produced by the combustion
model.EI.Streams.Dust           % Dust production resulting from the combustion



```

Listing A.19: Fuel Streams

Field	Var. type	Unit	Description
Type	cell string *		FUEL . Type of stream.
LHV	float	kJ/kg	Lower Heating Value.
AdiabaticTemp	float	K	Combustion adiabatic temperature
StackTemp	float	K	Minimum accepted temperature at the stack
State	int *		State of fuel (0 gas, 1 solid, 2 liquid).
Pressure	float	bar	Fuel pressure (required for a gaseous fuel).
Molw	float	kg/kmol	Fuel molar weight.
FlowMax	float	kg/s	Maximum fuel mass flow.
Cost	float	\$/kg	Fuel cost.
S	float	kg _S /kg _{Fuel}	Sulphur content of the fuel.
N	float	kg _N /kg _{Fuel}	Nitrogen content of the fuel.
O2	float	kg _{O2} /kg _{Fuel}	Oxygen mass flow consumed for a complete combustion.
CO2	float	kg _{CO2} /kg _{Fuel}	Mass of CO2 created for a complete combustion.
H2O	float	kg _{H2O} /kg _{Fuel}	Amount of water produced by the combustion
Dust	float	kg _{Dust} /kg _{Fuel}	Dust production resulting from the combustion.

Table A.20: Fuel fields from listing A.19

A.6.7 Steam - steam

Supported by:  and 

This particular stream is used to model the behavior of water and steam. The energy integration software automatically computes the stream shape in function of the stream state. Phase changes are automatically taken into account and the final stream may be cut in several parts. Listing A.20 shows the code (long and abridged definitions) to be used to define a **steam** stream. Table A.21 provides details about the fields. Fields from section A.2.2 are also available.


```

% Long definition
model.EI.Streams.Type = {'steam'}; % Type of stream.
model.EI.Streams.Flow % Mass flow [kg/s]
model.EI.Streams.Pressure % Pressure [bar]
model.EI.Streams.Tin % Inlet temperature [K] / vapour fraction [-]. Be
    careful with saturation points. Put 0.01 K more (or less) to ensure to be on the right
    side of the saturation curve.
model.EI.Streams.Tout % Outlet temperature [K] / vapour fraction [-]
model.EI.Streams.DTmin_2 %

% Abridged
model.EI.Streams.Short = {'steam','Parent','TagName',Flow,Pressure,Tin,Tout,DTmin_2} % Flow,
    Pressure,Tin,Tout and DTmin_2 can be numeric or strings

% Example of abridged definition
model(1).EI.Streams(3).Short = {'steam','myUnit','myStream',1,30,'443','0','myTag',3} % MyTag
    is a Tag coming for example from another model

```

Listing A.20: Steam Streams

Field	Var. type	Unit	Description
Type	cell string *		steam
Flow	float	kg/s	Mass flow.
Pressure	float	bar	Pressure.
Tin	float	K	Inlet temperature or vapour fraction.
Tout	float	K	Outlet temperature.
DTmin_2	float	ΔK	DTmin/2

Table A.21: steam fields from listing A.20

Note that both Tin and Tout fields can be used to set either temperatures or vapour fractions. In the value is set between 0 and 1, the software considers that a vapour fraction is set. The corresponding temperature is then automatically computed.

A.6.8 Variable Steam Streams - vsteam

These streams represent the thermodynamic behavior of water and steam. Change phase may take place between input and output temperature. Variable steam streams are similar to steam streams and they differ in the definition of the output temperature, which is defined within a discrete range instead of a fixed point. The energy integration software will chose the best output temperature depending of the integration problem to be solved.

vsteam streams are defined exactly as steam streams (see listing A.20, table A.21 and section A.6) plus some additional fields, described in listing A.21 and table A.22.



Field	Var. type	Unit	Description
FirstT	float	K	Lowest accepted output temperature.
LastT	float	K	Highest accepted temperature.
Prec	float	-	Discrete temperature interval between to possible output temperatures.


Table A.22: vsteam fields from listing A.21

```
% Example of vsteam stream
ns = ns+1;
model.EI.Streams(ns).Type = {'vsteam'};
model.EI.Streams(ns).TagName = {'vsteam_utility'};
model.EI.Streams(ns).DisplayName = {'vsteam stream'};
model.EI.Streams(ns).AddToProblem = 1;
model.EI.Streams(ns).Parent = {'cold'};
model.EI.Streams(ns).FirstT = 320;
model.EI.Streams(ns).LastT = 500;
model.EI.Streams(ns).Prec = 10;
model.EI.Streams(ns).Tin = 293;
model.EI.Streams(ns).Tout = 500;
model.EI.Streams(ns).Pressure = 1;
model.EI.Streams(ns).Flow = 3;
model.EI.Streams(ns).DTmin_2 = 5;
```

Listing A.21: vsteam stream

A.6.9 Common substances streams - `stream`

Supported by:  and 

EnergyTechnologies knows a certain number of substances and can reproduce their thermodynamic behavior. In this case, instead of using a `qt` stream, which requires to set two temperature-load couples for each state of the stream during a transformation, the user can use the `stream` stream. The materials accepted by EnergyTechnologies and supported by  are resumed in table A.23. Refer to the easy documentation (?) for more details about definition of these streams.

Category	Substances
Water	'EAU', 'EAUNBS', 'STEAM'
Fuels	'C2H4', 'C2H6', 'C3H8', 'C3H6', 'FUEL', 'GN', 'GN_DTG', 'OXYF'
Refrigerants	'NH3', 'R702', 'R720', 'R728', 'R740', 'R732', 'R50', 'R784', 'R704', 'R14', 'R1150', 'R170', 'R744', 'R23', 'R13', 'R116', 'R1270', 'R13B1', 'R290', 'R143', 'R717', 'R12', 'R22', 'R115', 'R218', 'R152', 'R40', 'R600A', 'R764', 'R630', 'R142', 'R600', 'R12B1', 'R160', 'R114', 'R21', 'R245FA', 'R245CA', 'RC318', 'PP50', 'PP5', 'R32', 'R134A', 'R125', 'R407C', 'R123'
Other	'AIR', 'FUMES'

Table A.23: Materials accepted by EnergyTechnologies

The syntax to be used to define a `stream` stream is given in listing A.23 and table A.24. Fields defined in section A.2.2 are also available.

In  only the substance 'AIR' is supported. Humid air streams can also be calculated.

When streams have more than one components, it is possible to impose the partial flow of some of the components. The others will be updated in order to maintain their relative concentration. This is useful for example to add the water flow to the stream air. For certain streams (see easy documentation), it is possible to specify the partial flowrate [kg/s] of the different substances composing the stream (N2, O2, H2O, ...). This allows one to modify the thermodynamic behavior of the stream. If this option is choosen, all the components must be defined. Adding the partial flow is done by adding to the material a string in brackets in the form "material (compound1=flow1;compound2=flow2)". flow are expressed in kg/s. The list of predefined material streams with more than 1 compound is given in table A.25. For

example, defining an air stream with a partial flow of air of 1.5 kg/s out of a total flow of 3.kg/s is defined by.

```
model.EI.Streams.Material='AIR (H2O=1.5)'           % Air with a partial flow of water of
1.5 [kg/s]
model.EI.Streams.Flow=3                             % Total Mass flow is 3 [kg/s]
% short version is
model.EI.Streams.Short = {'stream','AIR (H2O=1.5)','myUnit','myStream',3,1,'443','0','myTag',3}
% MyTag is a Tag coming for example from another model, total flow is 3 kg/s water
flow is 1.5 kg/s in the hot air flow
```

Listing A.22: Example of material definition

```
% Long definition
model.EI.Streams.Type = {'stream'}; % Type of stream.
model.EI.Streams.Material % Stream material
model.EI.Streams.Flow % Mass flow [kg/s]
model.EI.Streams.Pressure % Pressure [bar]
model.EI.Streams.Tin % Inlet temperature [K]. Be careful with saturation
points. Put 0.01 K more (or less) to ensure to be on the right side of the curve.
model.EI.Streams.Tout % Outlet temperature [K]
model.EI.Streams.DTmin_2 % [K]
model.EI.Streams.Components(...).TagName % Name of the component
model.EI.Streams.Components(...).Flow % Partial flowrate of the component
% Abridged
model.EI.Streams.Short = {'stream','material','Parent','TagName',Flow,Pressure,Tin,Tout,
DTmin_2} % Flow,Pressure,Tin,Tout and DTmin_2 can be numeric or strings
% Example of abridged definition
model(1).EI.Streams(3).Short = {'stream','AIR','myUnit','myStream',1,30,'443','myTag',3} %
MyTag is a Tag coming for example from another model
model(1).EI.Streams(3).Short = {'stream','AIR (H2O=1.5)','myUnit','myStream',3,30,'443','myTag'
,3} % MyTag is a Tag coming for example from another model, total flow is 3 kg/s water
flow is 1.5 kg/s in the hot air flow
```

Listing A.23: STREAM streams definition


Field	Var. type	Unit	Description
Type	cell string *		stream
Material	cell string *		Defines the material flow model. See tab. A.25 . The string may include the definition of the partial flows in brackets e.g. (H2O=1.5) to specify a partial flowrate of water of 1.5 kg/s in the flow.
Flow	float	kg/s	Mass flow.
Pressure	float	bar	Pressure.
Tin	float	K	Inlet temperature.
Tout	float	K	Outlet temperature.
DTmin_2	float	ΔK	DTmin/2
Components.TagName	cell string		Component identifier
Components.Flow	float	kg/s	Component partial flowrate

Table A.24: **stream** fields from listing [A.23](#)

Material	Component	Partial Flow
AIR	O2	0.210000
	N2	0.790000
	H2O	0.00000
FUMES	N2	0.00000
	O2	0.00000
	H2O	0.00000
	CO2	0.00000
GN	CH4	0.880000
	C2H6	0.900000E-01
	C3H8	0.200000E-01
	C4H10	0.500000E-02
R407C	N2	0.500000E-02
	R32	0.230000
	R125	0.520000
	R134A	0.250000

Table A.25: Material streams with multiple flows

A.6.10 Cost Streams - `cost`

Supported by: 

The structure shown in listing A.24 defines a generic cost stream. This stream requires two parameters `Cost1` and `Cost2`. Cost is computed as follows:

$$C = U_y \cdot c_1 + U_M \cdot c_2$$

where U_y is an integer value defining if the unit containing the stream is active or not and U_M is the unit multiplication factor (see process and utility units A.4.2 as examples). c_1 and c_2 define respectively `Cost1` and `Cost2`. The field `Cost` is added to the stream structure once its integration has been computed.

Cost streams are also used to compute investment cost and electrical power. These fields are converted from `Cinv1` and `Cinv2` (respectively `Power1` and `Power2`) fields defined in EI Units (sec. A.4.1).

Listing A.24 defines the syntax to be used to defined a `cost` stream. Generic fields defined in section A.6 are also available. Both explicit and short definitions are accepted.

```
% Long definition
model.EI.Streams.Type = {'cost'}; % Type of stream
model.EI.Streams.Layer % Name of the layer where the stream is considered
model.EI.Streams.Cost1 % Fixed cost
model.EI.Streams.Cost2 % Size-related cost

% Short definition
model.EI:Streams.Short = {'cost','layerName','parentName','tagName','Cost1','Cost2'};
```

Listing A.24: Cost stream

A.6.11 Mass Streams - `mass`

Supported by: 

This stream represents a mass stream, with a given flowrate. The structure shown in listing A.25 defines the syntax to be used to define mass streams. In addition to the generic data defined in section A.6, the syntax requires a flowrate and a direction, defining if the stream enters or leaves its unit. The definition of a **Layer** (see sec. A.3) is mandatory for mass streams, since no default layer is defined in the energy integration software.

Table A.18 explains the meaning of each field.

```
% Long definition
model.EI.Streams.Type = {'mass'}; % Type of stream.
model.EI.Streams.Layer % Parent layer, which must be of MassBalance Type.
model.EI.Streams.Parent % Parent Unit
model.EI.Streams.TagName % Stream name
model.EI.Streams.InOut % Flow direction
model.EI.Streams.Flowrate % Mass flowrate [kg/s]

% Abridged
model.EI.Streams.Short = {'mass','Layer','Parent','TagName','Direction','Flowrate'}

% Example of abridged definition
model(1).EI.Streams(3).Short = {'mass','coils','Pittsburg','coils_from_pitt','out',300};
```

Listing A.25: MASS stream

Field	Var. type	Unit	Description
Type	cell string *		mass
Layer	cell string		Parent layer, which must be of MassBalance Type.
InOut	cell string *		Flow direction. in , for streams entering the unit, out , for flows leaving the unit.
Flowrate	float	kg/s	Mass flowrate. Value must be positive.

Table A.26: **mass** fields from listing A.25

A.6.12 Mass Streams with quality - **mq**

Supported by: 

This stream represents a mass stream, with a given flowrate and a given quality. For example, this stream can represent hydrogen with a given purity or a fuel with a given LHV.

The syntax required to define a **mq** stream is identical to the syntax used to define **mass** streams (see listing A.25 and section A.6.11) with an additional **Quality** field defining the quality of the stream.

Listing A.26 shows an example of **mq** syntax. Like for **mass** streams, the definition of a **Layer** (see sec. A.3) is mandatory. For **mq** streams, layers must be of type **MassBalanceWithQuality**.

A.6.13 Mass Streams with partial flowrates - **mqpartial**

Supported by: 

This stream represents a mass stream, with one or more components defined by a given flowrate. For example, this stream can represent a natural gas stream, with given partial flowrates of methane, CO_2 , oxygen and so on. If the stream is an input of a unit, it is possible to specify a minimum partial flowrate of one or more components to be met after mixing all source streams.

The syntax required to define a **mqpartial** stream is similar to the syntax used to define **mass** streams (see

```

% Long definition
model.EI.Streams.Type = {'mq'};           % Type of stream.
model.EI.Streams.Layer = {'hydrogen'};    % Parent layer, which must be of MassBalance Type

model.EI.Streams.Parent = {'node1'};      % Parent Unit
model.EI.Streams.TagName = {'node1_out'}; % Stream name
model.EI.Streams.InOut = {'out'};         % Flow direction
model.EI.Streams.Flowrate = 12;           % Mass flowrate [kg/s]
model.EI.Streams.Quality = 0.95;          % Quality [-]

% Abridged
model.EI.Streams.Short = {'mq','Layer','Parent','TagName','Direction','Flowrate','Quality'}

% Example of abridged definition
model(1).EI.Streams(3).Short = {'mq','hydrogen','node1','node1_out','out',12,0.95};

```

Listing A.26: MQ stream

listing A.25 and section A.6.11) where the `Flowrate` field is replaced by a `Components` structure. Listing A.27 shows an example of `mqpartial` syntax, while table A.27 details `Components` fields. Like for mass streams, the definition of a `Layer` (see sec. A.3) is mandatory. For `mqpartial` streams, layers must be of type `MassBalanceWithQuality`. A global `Flowrate` is also accepted. No coherency check is performed between global and component-related flowrates.

```

s = struct;
s.Type = {'mqpartial'};
s.Layer = {'hydrogen'};
s.Parent = {'gary'};
s.TagName = {'gary_h2'};
s.InOut = {'in'};
s.Flowrate = 1000;
s.Components(1).TagName = {'h2'};
s.Components(1).Flowrate = 0.6;
s.Components(1).isConstrained = 1;
s.Components(2).TagName = {'co'};
s.Components(2).Flowrate = 0.3;
s.Components(3).TagName = {'ch4'};
s.Components(3).Flowrate = 0.1;
model.EI.Streams = osmose_append_structures(model.EI.Streams,s);


```

Listing A.27: MQPARTIAL stream

Field	Var. type	Unit	Description
TagName	cell string *		Component identifier. Must respect TagName specifications.
Flowrate	float		Component flowrate.
isConstrained	bool		Defines wheter the component partial flowrate must be satisfied. Only valid for <code>in</code> streams.

Table A.27: `Component` fields from listing A.27

A.7 EI Variables

Supported by: 

Variables can be defined in energy integration. They are not to be confused with *OSMOSE* variable (`o.Variables`), which are more general and valid for the whole model.

In energy integration, variables are often used in equations, to define constanst. But they can also be used to define real variables, which have to be computed by the energy integration software.

Listing A.28 and table A.28 define the syntax to be used to define a variable. Both long and abridged definitins are shown. Fields form section A.2.2 are also available.

```
% Long definition
model.EI.Variables.Status % Type of variable
model.EI.Variables.Value % Value of the variable
model.EI.Variables.Min % Minimum bound of the variable value
model.EI.Variables.Max % Maximum bound of the variable value
model.EI.Variables.Mult % Defines the multiplication factor of the variable
model.EI.Variables.Cost % Linear cost associated to the variable value

% Short definition
model.EI.Variables.Short = {'TagName','Parent','Status','Value','Min','Max','Mult','Cost'};
% Note that you can put NaN for field which are not mandatory depending on the variable
Status
```

Listing A.28: Energy Integration variables



Field	Var. type	Unit	Description
Status	cell string *		Indicated the variable status. See table A.29 for possible values.
Value	mixed		Indicates the value of the variable. May be float of int , depending on the variable Status .
Min	mixed		Minimum bound of the variable value. May be float of int , depending on the variable Status .
Max	mixed		Maximum bound of the variable value. May be float of int , depending on the variable Status .
Mult	float		Defines the multiplication factor of the variable.
Cost	float		Linear cost associated to the variable value.

Table A.28: Variables fields from listing A.28

Status	Requires	Description
cst	Value	The value is defined by the user. The variable becomes a constant.
off	Min , Max	The value is computed by the energy integration software. The value will be float .
int	Min , Max	The value is computed by the energy integration software. The value will be int .

Table A.29: Allowed Variables.Status

A.8 Constraints and Equations

Supported by:  and 

In the energy integration software it is possible to define constraints between streams. Constraints can concern multiplication factors, flows, integer variables and other parameters (see table A.30). The structure `EI.Equations` contains the definition of the equations, and the structure `EI.EqTerms` contain the terms of the equations.

Equations definition - `Equations` The `EI.Equations` field is used to define equations of the form

$$\sum_i c_i \cdot Term_i \left\{ \begin{array}{l} \leq \\ < \\ = \\ \neq \\ > \\ \geq \end{array} \right. RHT \quad (A.5)$$

The `EI.Equations` fields defines the name of the equation, its type, the right hand term (`RHT`) and the the (in)equality term, as shown in listing A.29 and table A.31. Fields from section A.2.2 are also available. The available types of equations are resumed in table A.30.









Type	Description
<code>mult</code>	The result of the equation is relative to the multiplication factor of the stream (or unit) concerned. Supports all units and streams.
<code>flow</code>	The result of the equation concerns the mass flow. Supported for <code>mass</code> and <code>mq</code> streams.
<code>integer</code>	The result of the equation concerns the integer variable (used for <code>SYNCRYC2</code> and <code>SYNUT2</code>). Supports all units and streams.
<code>heat</code>	The result of the equation is relative to the heat load of the streams. Supported for <code>qt</code> , <code>steam</code> , <code>vqt</code> and <code>vsteam</code> streams in  .
<code>power</code>	The result of the equation is relative to the electrical power of the or unit.
<code>cost</code>	 The result of the equation is relative to the operating cost of the stream or the unit (see sec. A.6.10).
<code>cin</code>	 The result of the equation is relative to the investment cost of the unit (see sec. A.6.10).
<code>quality</code>	 The results of the equation is relative to the quality of the <code>mq</code> stream (see sec. A.6.12)
<code>composition</code>	 The results of the equation is relative to the partial composition of one of the components of the <code>mqpartial</code> stream (see sec. A.6.13). An additional <code>Component</code> field is required in the <code>EqTerm</code> .
<code>partialflow</code>	 The results of the equation is relative to the partial flowrate of one of the components of the stream (see sec. A.6.13). An additional <code>Component</code> field is required in the <code>EqTerm</code> .
<code>mcp</code>	 The results of the equation is relative to the $\dot{M}c_p$ of the referred stream. <code>qt</code> and <code>vqt</code> streams are accepted.
<code>mix</code>	 The result of the equation is relative to a mix of the other elements listed this table. More specifications are required in <code>EqTerms</code> definition (see sec. A.8.1).

Table A.30: Equations types

When the equation is set to `mix`, equation terms must be defined with an additional `Type` field (see sec. A.8.1) defining which characteristic of the stream must be constrained.

```
model.EI.Equations.Type           % Defines the type of equation
model.EI.Equations.Equality       % ['<=', '=', '>', '>='] Defines the type of (in)equality
model.EI.Equations.RHT           % Right Hand Term value
```

Listing A.29: Matlab code for `Equations` field

Field	Var. type	Unit	Description
Type	cell string *		See table A.30.
Equality	cell string *		Defines the type of (in)equality. May be ' <code>=</code> ', ' <code>></code> ' (Easy) or ' <code>>=</code> ', or ' <code><=</code> ' (Exmpl).
RHT	float		Right Hand Term value.

Table A.31: `Equations` fields from listing A.29

A.8.1 Adding terms to the equations - `EqTerms`

Once one equation has been define, it is necessary to add at least one term. This can be done by using the `EI.EqTerms` field. To add one term, it is necessary to define its coefficient c_i , the name of the equation it belongs to and the name of the stream (or the variable (see sec. A.7) which the term is related to. Listing A.30 and table A.32 show the matlab code of the definition of the `EI.EqTerms` field. Fields from section A.2.2 are also available. Field `TagName` is mandatory.

Note that for each `EqTerm` only one of `StreamName` or `VarName` fields can be selected.

```
model.EI.EqTerms.StreamName       % Name of the stream concerned by the equation term
model.EI.EqTerms.VarName         % Name of the variable concerned by the equation term
model.EI.EqTerms.RefName         % Name of the entity concerned by the equation term:
    Replaces VarName and StreamName and supports Units
model.EI.EqTerms.Parent          % Name of the equation including the term
model.EI.EqTerms.Coefficient     % Value of the coefficient
model.EI.EqTerms.Type            % Type of equation term.
model.EI.EqTerms.Component       % Indicates the component for "component" equations
types
```

Listing A.30: Matlab code for `EqTerms` field



Field	Var. type	Unit	Description
StreamName	cell string		TagName of the stream concerned by the equation term.
VarName	cell string		TagName of the Ei variable concerned by the equation term.
RefName	cell string		 TagName of the Ei element concerned by the equation term. Replaces VarName and StreamName and supports Units.
Parent	cell string		TagName of the equation including the term.
Coefficient	float		Value of the coefficient.
Type	cell string		 Defines the characteristic of the stream concerned by the equation term. This field is required only when the equation is set to be mix . In this case, Type may take any value listed in table A.30 (excepted for mix).
Component	cell string		Indicates the stream component TagName (see sec. A.6.13) affected by the EqTerm.

Table A.32: EqTerms fields from listing A.30

When the equation term belongs to a **mix** equation (see sec. A.8), the field **Type** must be specified to define which characteristic of the stream must be constrained.

A.9 Heat exchange connection constraints

A.9.1 Heat exchange restrictions between units

Supported by: 

It is possible to add some specifications on heat exchange connections between two units or groups (mixed connection between a unit and a group is also possible). You can for example specify that a the connection between unit *UnitA* and *UnitB* is forbidden. To do this, the syntax used is presented on listing A.31 and table A.33. Fields from section A.2.2 are also available.

```

%% Syntax
model.EI.Connections.Type % Type of connection
model.EI.Connections.Mode % indicates the connection mode: mandatory, forbidden, ...
model.EI.Connections.From % Indicates the first element of the connection
model.EI.Connections.To % Indicates the second element (if any) of the connection

%% Example
% UnitA description
model.EI.Units(1).Type = {'process'};
model.EI.Units(1).TagName = {'UnitA'};
model.EI.Units(1).DisplayName = {'Unit A'};
model.EI.Units(1).AddToProblem = 1;
... %Description of other unit fields
model.Connections(1).Type = {'units'};
model.Connections(1).From = {'UnitA'};
model.Connections(1).To={'UnitB'}; % TagName of the unit with wich we want to specify
    the connection.
model.Connections(1).Mode={'0'}; % Mode of connection (cf. connection constraints
    table)
model.Connections(1).AddToProblem = 1; % Use the heat exchange connection specification

```

Listing A.31: Heat exchange connection between two units

Field	Var. type	Unit	Description
Type	cell string *		Indicates the type of connection. Only 'units' is available in EnergyTechnologies 2.0. In future versions, other types of connection may be implemented.
Mode	int *		Indicated the mode of connection. Table A.34 reports the available modes.
From	float		Indicates the source of the connection.
To	float		Indicates the target of the connection. This field is not necessary for Mode=1 .

Table A.33: Connections fields from listing A.31

Different connection modes are available. They are resumed in table A.34.

Mode	Description
0	Connections between From and To are forbidden.
1	Streams of From can only be connected with utility streams and inside From. In this case, definition of To in not required.
2	Connections only allowed between From and To and inside From or To .

Table A.34: Connections modes

A.9.2 Heat exchange restrictions between sub-systems

Supported by:  and 

It is also possible to devise industrial plants into sub-system with restricted heat exchange. Often heat exchange between sub systems are forbidden due to industrial constraints. Heat can be exchanged inside a sub-system but no direct heat exchange with other sub-systems is allowed. This constraint can be considered by simply adding one line to the unit or group definition to indicate that it should be treated as a sub-system (see listing A.32 and table A.35). There is no limit of number of sub-systems and every sub-system can be composed of units and/ or groups. A sub system has to be defined in a group.

In eiampL, for the moment only subsystem at one level are accepted.

To reduce energy penalties, introduced by industrial constraints, the integration of a heat transfer system (e.g. hot water loop or steam network) gives the possibility of indirect heat exchange between sub-systems.


```
%% Example
% GroupA sub-system description
model.EI.Groups(1).Type = {' '};
model.EI.Groups(1).TagName = {'GroupA'};
model.EI.Groups(1).DisplayName = {'Group A'};
model.EI.Groups(1).AddToProblem = 1;
model.EI.Groups(1).IsANode = {'1'}; % Indicates that GroupA is a subsystem
```

Listing A.32: Heat exchange connection between sub-systems

Field	Var. type	Unit	Description
IsANode	int *		Indicates if subsystem (IsANode = 1) or not (IsANode = 0)

Table A.35: IsANode field from listing A.32

A.10 Locations

Supported by: 

Locations define physical places where units are located. Locations are isolated, which means that no resources can be exchanged between locations, unless a distribution network is defined. This allow to configure problems where units are distant on an industrial site and direct heat or resources exchange is not feasible.

Locations are used in conjunction with **Networks** (see sec. [A.11](#)), in order to allow heat or resources transport between locations. In practice, this is done with a transfert media, like steam for heat.

If no **Location** is defined in the model description, a default location is loaded by osmose.

Listing [A.33](#) shows an example where two locations are defined. Listing [A.4](#) shows how the field **Location** is configured to define a location for a unit. Table [A.36](#) details the syntax.

```
nl=0;

nl=nl+1; model.EI.Locations(nl).TagName = {'loc1'};
model.EI.Locations(nl).x = {'583903.19'}; %x coordinate
model.EI.Locations(nl).y = {'147788.43'}; %y coordinate
nl=nl+1; model.EI.Locations(nl).TagName = {'loc2'};
```

Listing A.33: Definition of a Location

Field	Var. type	Unit	Description
TagName	cell string		Unique location identifier
x	cell string		coordinate
y	cell string		coordinate

Table A.36: Locations fields from listing [A.33](#)

A.11 Networks

TODO

A.12 Storage: SYNSTO

Supported by: 

SYNSTO unit is defined in order to integrate the storage tank in the multi period multi time optimization model. The following assumptions are considered to establish the storage model:

- Only liquid (sensible heat) storage will be considered. The heat load is therefore calculated considering a volume, a temperature difference and specific heat load.
- Only one storage tank could define in each location.
- The temperature in storage tanks is discretized into a finite number of temperature levels .

- The maximum storage tank volume (V_{\max}) that defines the amount of liquid needed in the storage system and the temperature discretization of the tanks are defined by the user (Max_T , Min_T and N_tanks).
- The approach can include an estimation of heat losses. They are proportional to the temperature and to the volume stored. The heat exchange area of the storage tank is calculated based on the predefined maximum volume capacity.

Heat losses is estimated based on the following equation:

$$\frac{k_{hl} \times f_{hl} \times 4 \times M_{s,r,p} \times (T_r - T_a)}{\rho \times d} \quad (\text{A.6})$$

Where, d shows the diameter of the tank, f_{hl} is a factor to account the heat losses on the top and bottom of the storage tank, k_{hl} is a heat loss coefficient, ρ is the density of the storage fluid and $M_{s,r,p}$ shows available quantity of fluid in each temperature intervals r and in each tank s during time step p .

Listing A.34 shows an example where a storage unit is defined. Table A.37 details the syntax.

```

nu=0;

nu = nu+1;
model.EI.Units(nu).Type = {'SYNSTO'};
model.EI.Units(nu).TagName = {'storage1'};
model.EI.Units(nu).DisplayName = {'storage unit'};
model.EI.Units(nu).AddToProblem = {'1'};
model.EI.Units(nu).Fmin = {'0'};%
model.EI.Units(nu).Fmax = {'100'};
model.EI.Units(nu).Cost1 = {'0'};
model.EI.Units(nu).Cost2 = {'0'};
model.EI.Units(nu).Cinv1 = {'0'};
model.EI.Units(nu).Cinv2 = {'0'};
model.EI.Units(nu).Power1 = {'0'};
model.EI.Units(nu).Power2 = {'0'};
model.EI.Units(nu).Max_T = {'130'};
model.EI.Units(nu).Min_T = {'90'};
model.EI.Units(nu).N_tanks = {'1'};
model.EI.Units(nu).CP = {'4.18'};% [MJ/kg k]
model.EI.Units(nu).Flow = {'1'};% [kg/s]
model.EI.Units(nu).Rho = {'1000'};% density in kg/m3
model.EI.Units(nu).Vmax = {'100'};% total volume of all tanks in m3
model.EI.Units(nu).Location = {'loc1'};
model.EI.Units(nu).Parent= {'storage_g'};
model.EI.Units(nu).HeatLosses = {'1'};% 0 no losses, 1 considers heat losses by estimation
model.EI.Units(nu).TankDiameter = {'3'};% The diameter of a tank [m]
model.EI.Units(nu).HLCoefficient = {'1/1000'};% [kW/m^2 k] heat loss coefficient
model.EI.Units(nu).HLF = {'1.1'};% [-] is factor to account heat losses on the top and
bottom of the storage tank


```

Listing A.34: Definition of a storage

Field	Var. type	Unit	Description
Max_T	cell string	C	Maximum temperature in the tank
Min_T	cell string	C	Minimum temperature in the tank
N_tanks	cell string	-	Number of tank in one storage unit
CP	cell string	MJkg k	
Rho	cell string	kg/m ³	density
Vmax	cell string	m ³	Maximum volume of the tank
TankDiameter	cell string	m	The diameter of a tank
HLCoefficient	cell string	kW/m ² k	heat loss coefficient
HLF	cell string	-	It is factor to account heat losses on the top and bottom of the storage tank

Table A.37: Storage fields from listing A.34

A.13 Multi time, Multi periods

Supported by: 

There is a possibility of defining sequential time steps in each period by using Multi period-Multi time definition. It is refer to ? for more information.

In osmose tags' values of each time step are defined as constants in the frontend. In order to make it straightforward you could use separate file like *function o = data_file(o)* to define tags' values in each time step and then call it like *o = data_file(o)* in the frontend.

Listing A.35 shows an example where a data file is defined. Table A.38 details the syntax.

```
function o = data_file(o)

i=0; % Number of time depending TagName

i = i+1;
o.Constants(i).ModelTagName = {'modelName'}; % Model name
o.Constants(i).TagName = {'TagName1'}; % TagName
o.Constants(i).Period(1).Time(1).Value =13;
o.Constants(i).Period(1).Time(2).Value =15;
o.Constants(i).Period(2).Time(1).Value =14;
o.Constants(i).Period(2).Time(2).Value =16;

i = i+1;
o.Constants(i).ModelTagName = {'modelName'};
o.Constants(i).TagName = {'TagName2'};
o.Constants(i).Period(1).Time(1).Value =37;
o.Constants(i).Period(1).Time(2).Value =42;
o.Constants(i).Period(2).Time(1).Value =39;
o.Constants(i).Period(2).Time(2).Value =40;
```

Listing A.35: Definition of a data file in Multi Time-Multi Period

Field	Var. type	Unit	Description
ModelTagName	cell string		TagName of the model
TagName	cell string		Time dependent TagName
Value			Tag's value in each time step

Table A.38: Data file fields from listing [A.35](#)

Appendix B

Emissions

TODO: this chapter is a stub. The method still must be developed.

Emissions are computed on nominal conditions, corresponding to the size provided as input in the database call.

Following emissions are computed:

- CO_2 , **C02**, [g/s]
- NO_x , **N0x**, [g/s]
- SO_2 , **S02**, [g/s]
- CO , **C0**, [g/s]
- HC , **HC**, [g/s], hydro-carbons
- Particulates, **Particulates**, [g/s]

Listing [B.1](#) shows the matlab structure **Emissions** containing the information about emissions.

```
model.Emissions.C02           % [g/s] : C02 concentration in the exhaust gas
model.Emissions.N0x           % [g/s] : N0x concentration in the exhaust gas
model.Emissions.S02           % [g/s] : S02 concentration in the exhaust gas
model.Emissions.C0            % [g/s] : C0 concentration in the exhaust gas
model.Emissions.HC            % [g/s] : HC concentration in the exhaust gas
model.Emissions.Particulates  % [g/s] : Particulate concentration in the exhaust
gas
```

Listing B.1: Emissions structure

Appendix C

Costs

TODO: this chapter is a stub. The method must still be developed. Some code is already available on svn at <https://leni.epfl.ch/svn/EquipmentCost/trunk/>. A documentation is available at the same url.

The database provides several information about equipment cost. The values are recovered from the Equipment Cost database developed at LENI.
The following costing structure is provided:

Reference year `Reference_year`, [-] Costing relationships are computed for a particular year. Using Marshall&Swift costing index, it is possible to actualize the cost of the equipments.

Marshall&Swift Index, `MS_Index` Marshall&Swift Costing Index

Bare Module Cost, `BM`, [€ | \$ | CHF] Direct and indirect expenses related to the unit: material, construction, freight, taxes, overheat, engineering expenses, installation.

Total Module Cost, `TM`, [€ | \$ | CHF] Bare module cost plus the contingencies and fees due to unforeseen circumstances and fees during the plant commissioning.

Specific Grass Root, `SpecificGR`, [€ | \$ | CHF] Sum of the total module cost plus the cost for auxiliary facilities. This cost corresponds to the overall capital investment of a new plant without considering the expenses for land.

Unit, `Unit`, [€ | \$ | CHF] Currency of the computed cost.

Maintenance, `Maintenance`, [€ | \$ | CHF]

Operation, `Operation`, [€ | \$ | CHF] Including hours of work,...

α_1 , `f_TotalModule`, [-] Factor used to consider contingencies and fees in Total Module Cost.

α_2 , `f_GrassRoot`, [-] Factor used to consider construction of auxiliary facilities in Grass Root Cost.

```

%% Costing
model.Cost.Reference_year % [yyyy] : Reference year for M&S computation
model.Cost.MS_Index % [-] : Marshall&Swift costing index
model.Cost.BM % [euro|$|CHF] : Bare Module Cost
model.Cost.TM % [euro|$|CHF] : Total Module Cost
model.Cost.SpecificGR % [euro|$|CHF] : Specific GrassRoot
model.Cost.Maintenance % [euro|$|CHF] : maintenance cost
model.Cost.Operation % [euro|$|CHF] : operation cost
model.Cost.f_TotalModule % [-] : contingencies and fees factor
model.Cost.f_GrassRoot % [-] : auxiliary facilities factor
model.Cost.Unit % [euro|$|CHF] : Currency

```

Listing C.1: Cost structure

Appendix D

Life Cycle Assessment

D.1 What is Life Cycle Assessment?

Life cycle assessment (LCA) is a methodology used to assess the environmental impacts of a product, a service or a system, taking into account its overall life cycle and related to its function. It has been extensively used to identify the possible points of improvement for a given product, compare different products or systems, or compare the different stages of the life cycle of a particular product. LCA is submitted to ISO norms 14'040 & 14'044, and is effectuated in four main stages: *the goal and scope definition, the life cycle inventory, the impact assessment and the interpretation*. Figure D.1 summarizes the four stages of the LCA and shows also how they have to be conducted to build a LCA model in Energy Technologies. In dashed are the steps that do not change compared to a conventional LCA.

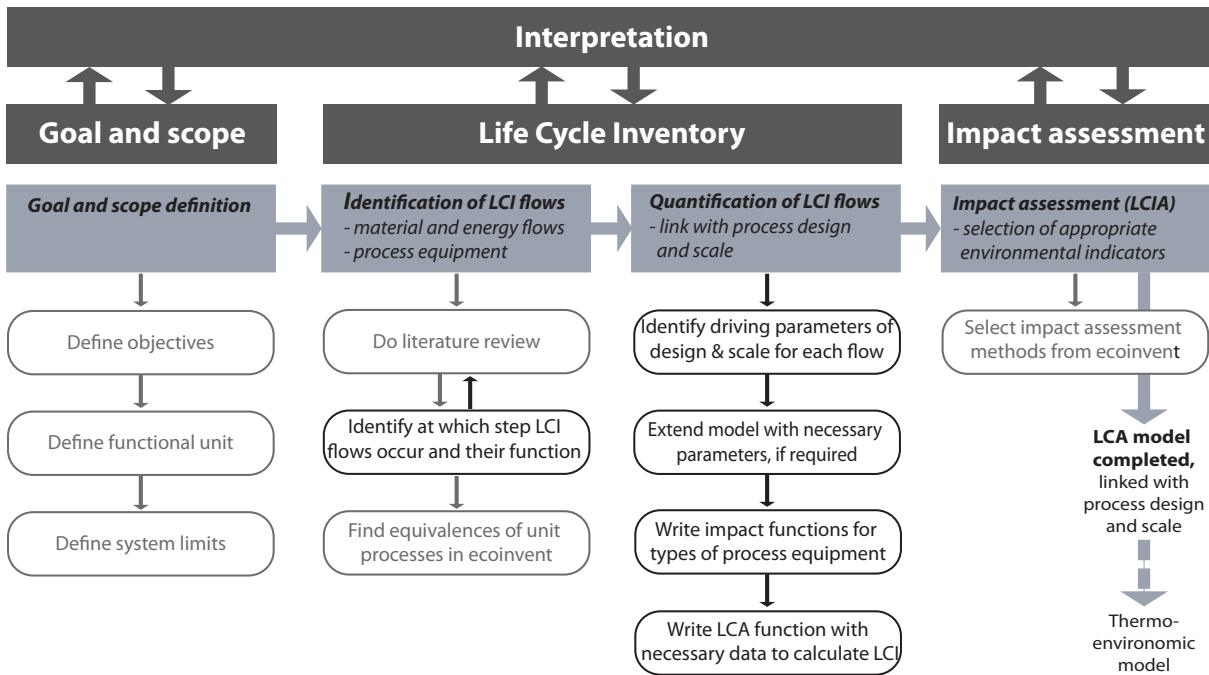


Figure D.1: representation of the major steps to be conducted in a LCA

At the first stage of a LCA, the *goal and scope definition*, the function of the studied system is defined, from which the functional unit (FU) will be determined. FU is a value used to quantify the function of the system, and every flow falling within the system boundaries is to be brought back to this FU. Then, system boundaries are defined with respect to the function of the system, and the system can be

represented as a sequence of interconnected unit processes, as shown at figure D.2.

At the second phase of the LCA, the *life cycle inventory*, every flow crossing the system boundaries as an extraction or an emission which is necessary to one of the unit processes is identified and quantified. These elements form the emissions and extraction inventory or life cycle inventory (LCI), and can be expressed as a vector of cumulated single substances, or elementary flows. This is illustrated by figure D.2.

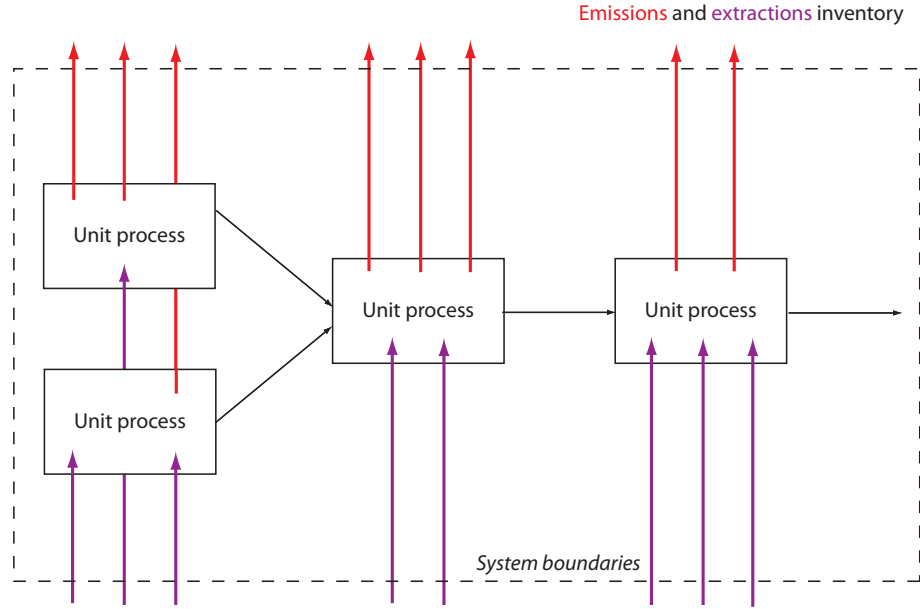


Figure D.2: schematic representation of the construction of a life cycle inventory

Generally, a database of emissions and extraction inventories for different unit processes is used at this step. These datasets have their own system limits for the life cycle analysis of the unit processes, including upstream unit processes and sometimes downstream processes such as waste disposal. The use of a database allows therefore expressing the LCI as a sequence of unit processes with exhaustive and comparable extraction and emissions inventories for each unit processes, but in such a case it is recommended to check carefully the subsystem limits for each unit process. The use of such a database is possible in Energy Technologies. For the moment, the ecoinvent database¹ is used in Energy Technologies. In the case the user wants to use another database, it is not recommended to mix the unit processes of different databases, since the list of elementary flows taken into account in the emissions and extraction inventory might not be the same, and therefore the different unit processes not comparable among them.

At the third phase of a LCA, the *life cycle impact assessment*, the cumulated emissions and extraction inventory, forming a substance vector, is aggregated in more global indicators, having an environmental significance. This is done by the mean of a matrix multiplication, where each elementary flow of the substance vector is assigned an impact factor by an impact matrix. This is illustrated by equation D.1.

$$\begin{bmatrix} F_{1,1} & \dots & F_{1,n} \\ \dots & \dots & \dots \\ F_{m,1} & \dots & F_{m,n} \end{bmatrix} * \begin{bmatrix} E_1 \\ \dots \\ E_n \end{bmatrix} = \begin{bmatrix} I_1 \\ \dots \\ I_m \end{bmatrix} \quad (\text{D.1})$$

The fourth phase of the LCA is *interpretation*, which goal is to summarize the results and to bring useful information for decision making, is performed after the life cycle inventory or after the life cycle impact assessment.

¹<http://www.ecoinvent.org>

In Energy Technologies, the life cycle inventory phase is conducted with more precise guidelines than the ones provided by the general LCA methodology, in order to adapt LCI to process design and scale. It is parsed in two sub-phases, as shown by figure D.1 , which are the LCI elements identification, where all elements are precisely identified, and LCI quantification, where mathematical expressions are developed to adapt these LCI elements to design and scale.

More information on conventional LCA and on how to conduct each one of the steps is available in *Analyse du cycle de vie, Comprendre et réaliser un ecobilan* by Olivier Jolliet, Myriam Saade, Pierre Crettaz, edited at the *Presses Polytechniques et Universitaires Romandes*.

D.1.1 Definitions

Table D.1 report the different technical terms related to LCA and their associated definitions, and can be used as a reminder. Some of the terms are generic terms used in the field of LCA, some of them have been specifically created for the application of LCA to Energy Technologies.

Term	Abbreviation	Definition
Component	-	Piece of process equipment or infrastructure, with impact following a non-linear law linked to a functional parameter
Elementary Flow	EFlow	Single emission or extraction, can also define a LCI element
Emissions and extractions inventory	-	Description of the different flows of pollutants, energy, raw materials, crossing the system boundaries, regrouping inlet flows that represent extracted resources and outlet flows that are pollutants emitted, under the form of elementary flows
Functional parameter	-	Parameter linked to the function of a particular type of process equipment, function of the size of this process equipments, allows to perform the impact scaling
Functional unit	FU	Value used to quantify the function of the system, to which every flow is brought back, used to compare different systems or scenarios
Goal and Scope Definition	-	1st stage of LCA, describing the objectives, defining the function of the studied system and the functional unit, and the system limits
Impact Assessment Method	-	Method modeling the impact pathways of the different elementary flows and linking them to their environmental impact
Impact category	-	Category grouping the elements of the substance vector after weighing by impact matrix, can express either a similar effect on the environment or a damage, depending on the impact assessment methods
Impact matrix	-	Matrix containing the impact factors, or weights, of all elementary flows, used to translate the substance vector into a vector of impact categories
Interpretation	-	4th phase of LCA, summarizing the results of the 3 other phases and providing useful information for decision making
Life Cycle Assessment	LCA	Method to evaluate the environmental impact of a product, a system or a service, with respect to its function and taking into account every step of its life cycle
Life Cycle Impact Assessment	LCIA	3rd stage of LCA, linking emissions and extractions inventory of elementary flows to their impact on the environment
Life Cycle Inventory	LCI	2nd phase of LCA, quantifying the different flows of unit processes crossing the system boundaries
Unit Process	Unit Process	Emissions and extraction inventory for a given LCI element, cumulating the life cycle steps for a defined subsystem for this LCI element
Substance vector	-	Vector containing all elementary flows of cumulated emissions and extractions inventory
System boundaries	-	Delimitations to consider for modeling of the studied system, including all processes useful to its function

Table D.1: Definitions of the terms used in LCA

D.2 Specificities of LCA for components of process equipment

This section may be useful to anyone who wants to make a detailed LCA of the pieces of process equipment accounted for in a LCI, using the parameters of lifetime, maintenance, recycling and disposal. The impact of a single component (e.g. piece of process equipment) is performed for its overall life cycle, which system limits and stages are described in Figure D.3.

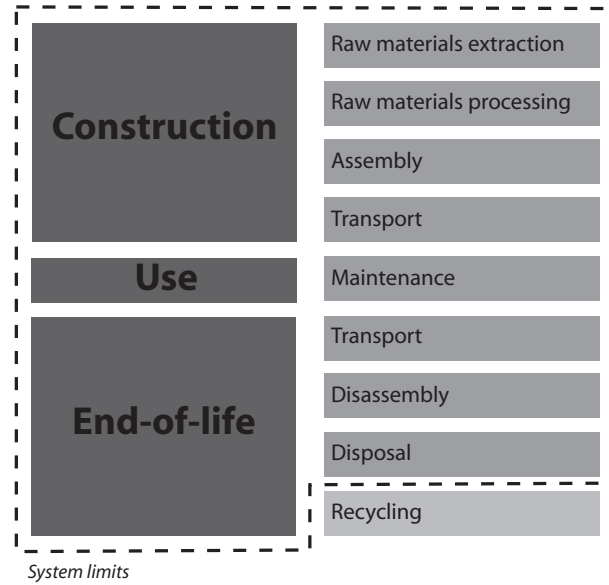


Figure D.3: schematic representation of the life cycle of a component.

In addition to the impact of the component production itself, which is linked to the size of the component itself, other impacts are generated during the lifecycle, due to the transports from the production place to the use place, to the maintenance, to the transport for disposal of process equipment at end-of-life and to its disposal. Part of the process equipment may be recycled, and in this case the impacts of the recycling are not accounted for since the recycled quantity of material is leaving the system. Therefore, once the impact of the production of the component is known, the total life cycle impact of the process equipment may be expressed as:

$$I_{c,tot} = (I_{c,p} + I_{c,d}(d, m, type_d) + I_{c,w}(m, r, type_w)) \cdot \tau t \quad (D.2)$$

where $I_{c,tot}$ is the total impact of the component for its overall lifecycle, $I_{c,p}$ is the impact due to the production of the component after its assembly, $I_{c,d}$ is the impact due to the transport of the component between the assembly and the use phase, and between the use phase and the disassembly, $I_{c,w}$ is the impact of the component due to its disassembly and disposal, t is the lifetime of the installation and τ is the maintenance time ratio, which is the percentage of the life cycle component impact that has to be added for each year of lifetime of the installation, accounting thus for the maintenance of the components. $I_{c,d}$ is function of the total transport distance d , of the mass m of the component, and of the type of transport $type_d$ (i.e. lorry, rail,...). $I_{c,w}$ is function of m , of the recycling ratio for the component r , and of the type of disposal $type_w$ (i.e. landfill, incineration,...).

Default values are assumed for these parameters, and they should be changed in the case the user has a very specific and detailed problem to solve where the life cycle impact of the process equipment has to be studied in details.

The implementation of these parameters in EnergyTechnologies are described at section D.7.1 and D.8.

Regarding the calculation of the impact of the production of the component itself, this is beyond the scope of this documentation. More details can be found in *Gerber L, Gassner M, Marechal F.*

Integration of LCA in the optimal design of energy conversion systems: Application to SNG production from lignocellulosic biomass. Computers & Chemical Engineering, 2010.

D.3 Multi-period and Life Cycle Stage modes

This section may be useful to anyone who wishes to use the LCA module in multi-period mode in a life cycle perspective. It presents the theoretical statements to be known before declaring the LCI elements of the models, and the way the LCA calculations are performed within osmose in life cycle stage mode.

D.3.1 Definition of the life cycle stages

The life cycle of a product, a system or a service has usually three distinct successive phases: *the construction phases*, *the use phase* and *the end-of-life phase*, which can be accounted or not in the system boundaries, depending on the goal and scope of the LCA. This is summarized by Figure D.4.



Figure D.4: schematic representation of the different life cycle stages of a product, system or service

In a mono-period problem, this does not involve to make any special between these three stages, which can be all assessed in a similar way. However, in a multi-period problem, this is not the case, since the construction phase and the end-of-life phase may depend on the different periods of the use phase, like the maximal investment linked with an energy system may depend on the different periods of operation. This principle is explained in Figure D.5.

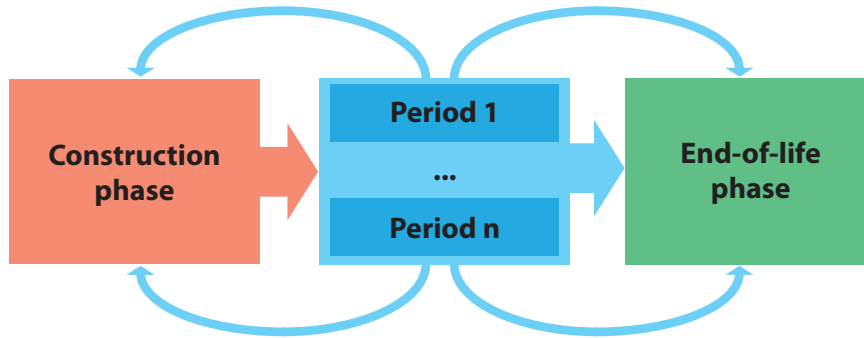


Figure D.5: schematic representation of the life cycle stages and of their interactions in a multi-period problem

Thus, the total impact of the system can be calculated by Equation D.3:

$$I_{tot} = \sum_{p=1}^{n_p} \sum_{i=1}^{n_{eo}} I_{O_{i,p}} + \sum_{i=1}^{n_{ec}} \max(I_{C_i})_p + \sum_{i=1}^{n_{ee}} \max(I_{E_i})_p \quad (D.3)$$

where I_{tot} is the total impact of the system, n_p is the number of periods considered in the problem, n_{ec} is the number of LCI elements belonging to the construction phase, n_{eo} is the number of LCI elements belonging to the operation phase, n_{ee} is the number of LCI elements belonging to the end-of-life phase, I_C is the impact of the construction phase, I_O is the impact of the operation phase, I_E is the impact of the end-of-life phase.

D.4 Method interface syntax

The method requires components, cumulative flows or elementary flows to be defined in order to run a computation. It is not an obligation that each type of LCI elements is defined within the model. Some other specific LCA parameters are also required to run a computation.

The LCIA interface uses the `model.LCIA` structure. This structure is then parsed between `model.LCIA.LCI`, which contains the LCI elements, and `model.LCIA.Params`, which contains the parameters. The first-level fields and second-level fields of this structure are shown in listing D.1. Details about the content for the `model.LCIA.LCI` structure are given in the following paragraphs.

```
model.LCIA.LCI          % LCI elements definition
model.LCIA.LCI.UProcess % Unit process definition
model.LCIA.LCI.EFlows   % Elementary flows definition
model.LCIA.LCI.Components % Components definition
```

Listing D.1: Content of the `model.LCIA.LCI` structure

D.4.1 Generic fields for all the syntax elements of LCI

Fields reported in listing D.2 and table D.2 are common to all LCI elements: `Components`, `UProcess`, `EFlows`. They allow to identify the LCI element and to configure the usage inside the problem (`AddToProblem`).

```
% LCI element identification
model.LCIA.LCI(...).TagName % LCI element identifier
model.LCIA.LCI(...).DisplayName % Short name of the LCI element. Used for display
purposes.
model.LCIA.LCI(...).Description % Long description of the LCI element.
model.LCIA.LCI(...).AddToProblem % Defines if the LCI element shall be added to the
problem.
```

Listing D.2: Generic fields for LCI elements

Field	Var. type	Unit	Description
<code>TagName</code>	<code>cell string</code>		LCI element identifier.
<code>DisplayName</code>	<code>cell string</code>		Short name of the LCI element. Used for display purposes.
<code>Description</code>	<code>cell string</code>		Long description of the LCI element.
<code>AddToProblem</code>	<code>bool</code>	-	Defines if the LCI element shall be added to the problem. <code>0</code> allows to deactivate the entity, while <code>1</code> enables it.

Table D.2: Generic fields for LCI elements from listing D.2

D.5 Unit Processes

A unit process relates to a dataset containing extensive life cycle inventories data, cumulating the emissions and extractions for a predefined system. For the moment, the life cycle inventories database used

for unit processes is ecoinvent, and files for many different unit processes can be downloaded and used, either in .xml or .mat format. The information relative to the unit processes is contained in the fields of the `model.LCIA.LCI.UProcess` structure.

D.5.1 Generic fields for unit processes

Listing D.3 and table D.3 report fields that are used for the definition of unit processes. Fields defined in section D.4.1 are also available.

```
model.LCIA.LCI.UProcess.UProcessID % ecoinvent ID of the corresponding unit process
    dataset equivalent to the unit process.
model.LCIA.LCI.UProcess.Value      % quantity of the unit process
model.LCIA.LCI.UProcess.Unit       % unit in which the quantity is given
model.LCIA.LCI.UProcess.LifeCycleStage % life cycle stage to which the unit process
    belongs
```

Listing D.3: Unit processes

Field	Var. type	Unit	Description
<code>UProcessID</code>	cell string		ecoinvent ID of the corresponding unit process dataset equivalent to the unit process
<code>Value</code>	float		quantity of the unit process
<code>Unit</code>	cell string		unit in which the quantity is given
<code>LifeCycleStage</code>	cell string		code for the life cycle stage ('C'=construction, 'O'=operation(use), 'E'=end-of-life)

Table D.3: Generic unit processes fields from listing D.3

The 'Unit' field has to be the unit imposed by the datasets of ecoinvent, since for the moment there is no automatic conversion of the units in EnergyTechnologies.

D.5.2 Integrating new datasets

Since the ecoinvent database is being updated and completed with time, it can happen that a dataset available in the online database is not available in the svn version of the database. In the case, the dataset can be downloaded from the ecoinvent website and added to the svn database. The procedure is the following:

1. When you are on the page describing the dataset you want to download, click 'Add to download-basket'.
2. Go to 'Show download basket'.
3. Click 'Download marked datasets'.
4. If not selected, select the options 'LCI results' and 'Without' impact assessment.
5. Click 'Generate EcoSpold Data'.
6. Click '1 Process' to the left of the dataset description.
7. Save your .xml file under the 'uprocess' folder of your ecoinvent database downloaded from svn.
8. Rename your file with a new 5 number-ID, preferably the one used in the online version of ecoinvent. The file can now be called within any of your EnergyTechnologies models, using the 'UProcessID' field.

D.6 Elementary Flows

An elementary flow relates to a single extraction or emission, that is added to the life cycle inventory of the system. For the moment, the used database is ecoinvent, which lists many different possible elementary flows. The information relative to the elementary flows is contained in the `model.LCIA.LCI.EFlows` structure.

D.6.1 Generic fields for elementary flows

Listing D.4 and table D.4 report fields that are used for the definition of elementary flows. Fields defined in section D.4.1 are also available.

```
model.LCIA.LCI.EFlows.EFlowID % ecoinvent ID of the corresponding elementary flow in the
                               LCI vector and impact matrix
model.LCIA.LCI.EFlows.Value   % quantity of the elementary flow
model.LCIA.LCI.EFlows.Unit    % unit in which the quantity is given
model.LCIA.LCI.EFlows.LifeCycleStage % life cycle stage to which the elementary flow
                                   belongs
```

Listing D.4: Elementary Flow

Field	Var. type	Unit	Description
<code>EFlowID</code>	<code>int</code>	-	ecoinvent ID of the corresponding elementary flow in the LCI vector and impact matrix
<code>Value</code>	<code>float</code>		quantity of the elementary flow
<code>Unit</code>	<code>cell string</code>		unit in which the quantity is given
<code>LifeCycleStage</code>	<code>cell string</code>		code for the life cycle stage ('C'=construction, 'O'=operation(use), 'E'=end-of-life)

Table D.4: Generic elementary flows fields from listing D.4

D.7 Components

A component relates to a piece of process equipment which is required in the infrastructure of the system. The impact of components is non-linear, oppositely to the cumulative and elementary flows, and scaling of the life cycle inventory using a more complex mathematical law is therefore required. This mathematical law uses one or more functional parameters related to the size and the operating conditions of the process equipment to perform the scaling. The information relative to the components is contained in the `model.LCIA.LCI.Components`.

D.7.1 Generic fields for components

Listing D.5 and table D.5 report fields that are used for the definition of components. Fields defined in section D.4.1 are also available.

```

model.LCIA.LCI.Components.Type % type corresponding to the component.
model.LCIA.LCI.Components.Data % structure to define the value of the necessary data
model.LCIA.LCI.Components.Unit % structure to define the units in which the data are
    defined
model.LCIA.LCI.Components.Lifetime % (optional) lifetime of process equipment
model.LCIA.LCI.Components.MaintenanceTimeRatio % (optional) yearly percentage of impact
    due to maintenance
model.LCIA.LCI.Components.TransportDistance % (optional) distance of transport after
    production of components, during other phases of life cycles
model.LCIA.LCI.Components.RecyclingRatio % (optional) percentage of component recycled
    at end-of-life, leaving the system boundaries
model.LCIA.LCI.Components.TransportID % (optional) equivalence for life cycle transport
model.LCIA.LCI.Components.DisposalID % (optional) equivalence for disposal (part
    not recycled)

```

Listing D.5: Components

Field	Var. Type	Unit	Description
Type	cell string		type corresponding to the component
Data	float		structure to define the value of the necessary data
Unit	cell string		structure to define the units in which the data are defined
Lifetime	float	years	lifetime of process equipment
MaintenanceTimeRatio	float	-	yearly percentage of impact due to maintenance
TransportDistance	float	km	distance of transport after production of components, during other phases of life cycles
RecyclingRatio	float	-	percentage of component recycled at end-of-life, leaving the system boundaries
TransportID	cell string		equivalence for life cycle transport, unit process ID
DisposalID	cell string		equivalence for disposal (part not recycled), unit process ID

Table D.5: Generic components fields from listing D.5

The different types of components that are available are listed in table D.6.

Type	Description
Boiler	Defines a boiler
Compressor	Defines a compressor, axial or centrifugal
Engine	Defines an engine, for cogeneration or single electricity production
Filter	Defines a bag filter
FlashDrum	Defines a flash drum
HeatEx_ST	Defines a shell and tube heat exchanger, tubes being in stainless steel and shell being in unalloyed steel
Molecular_Sieve	Defines a molecular sieve filled with zeolite
Pump	Defines a pump
Reactor_SS	Defines a reactor, in stainless steel
Tank_Concrete	Defines a tank, in concrete
Turbine	Defines a turbine, axial

Table D.6: Types of components available

Data that need to be defined for each type of unit are described in the following sections.

D.7.2 Boiler

Listing D.6 and table D.7 report fields that are used to define a boiler. The life cycle inventory and the impact of the boiler are scaled with respect to the thermal power. Fields defined at sections D.4.1 and D.7.1 are also available.

```
model.LCIA.LCI.Components.Data.Power    % thermal power definition
model.LCIA.LCI.Components.Units.Power   % units in which the thermal power is defined
```

Listing D.6: Boiler

Field	Var. Type	Unit	Description
Data.Power	float	kW	thermal power definition
Units.Power	cell string		units in which the thermal power is given

Table D.7: Generic boiler fields from listing D.6

D.7.3 Compressor

Listing D.7 and table D.8 report fields that are used to define a compressor. The life cycle inventory and the impact of the compressor are scaled with respect to the electrical power. Fields defined at sections D.4.1 and D.7.1 are also available.

```
model.LCIA.LCI.Components.Data.Power    % electrical power definition
model.LCIA.LCI.Components.Units.Power   % units in which the electrical power is defined
```

Listing D.7: Compressor

Field	Var. Type	Unit	Description
Data.Power	float	kW	electrical power definition
Units.Power	cell string		units in which the electrical power is given

Table D.8: Generic compressor fields from listing D.7

D.7.4 Engine

Listing D.8 and table D.9 report fields that are used to define an engine. The life cycle inventory and the impact of the engine are scaled with respect to the electrical power, and if it is used for cogeneration or not. Electrical efficiency is used to calculate the mass of the component. Fields defined at sections D.4.1 and D.7.1 are also available.

Field	Var. Type	Unit	Description
Data.Power	float	kW	electrical power definition
Units.Power	cell string		units in which the electrical power is given
Data.Cogen	float	-	used for cogeneration (1) or not (0)
Data.ElecEff	float	-	electrical efficiency definition
Units.ElecEff	cell string		units in which the electrical efficiency is given

Table D.9: Generic engine fields from listing D.8

```

model.LCIA.LCI.Components.Data.Power      % electrical power definition
model.LCIA.LCI.Components.Units.Power     % units in which the electrical power is defined
model.LCIA.LCI.Components.Data.Cogen      % whether used for cogeneration (1) or not (0)
model.LCIA.LCI.Components.Data.ElecEff    % electrical efficiency definition
model.LCIA.LCI.Components.Units.ElecEff   % units in which the electrical efficiency is
defined

```

Listing D.8: Engine

D.7.5 Filter

Listing D.9 and table D.10 report fields that are used to define a filter. The life cycle inventory and the impact of the filter are scaled with respect to the volumetric flow rate. Fields defined at sections D.4.1 and D.7.1 are also available.

```

model.LCIA.LCI.Components.Data.VolumetricFlowRate % volumetric flow rate definition
model.LCIA.LCI.Components.Units.VolumetricFlowRate % units in which the volumetric flow
rate is defined

```

Listing D.9: Filter

Field	Var. Type	Unit	Description
Data.VolumetricFlowRate	float	m ³ /s	volumetric flow rate definition
Units.VolumetricFlowRate	cell string		units in which the volumetric flow rate is given

Table D.10: Generic filter fields from listing D.9

D.7.6 FlashDrum

Listing D.10 and table D.11 report fields that are used to define a flash drum. The life cycle inventory and the impact of the flash drum are scaled with respect to the volumetric flow rate and the pressure. Fields defined at sections D.4.1 and D.7.1 are also available.

```

model.LCIA.LCI.Components.Data.VolumetricFlowRate % volumetric flow rate definition
model.LCIA.LCI.Components.Units.VolumetricFlowRate % units in which the volumetric flow
rate is defined
model.LCIA.LCI.Components.Data.Pressure          % pressure definition
model.LCIA.LCI.Components.Units.Pressure          % units in which the pressure is defined

```

Listing D.10: Flash Drum

Field	Var. Type	Unit	Description
Data.VolumetricFlowRate	float	m ³ /s	volumetric flow rate definition
Units.VolumetricFlowRate	cell string		units in which the volumetric flow rate is given
Data.Pressure	float	bar	pressure definition
Units.Pressure	cell string		units in which the pressure is given

Table D.11: Generic filter fields from listing D.10

D.7.7 HeatEx_ST

Listing D.11 and table D.12 report fields that are used to define a tube and shell heat exchanger. The life cycle inventory and the impact of the heat exchanger are scaled with respect to the area and the operating pressure. Fields defined at sections D.4.1 and D.7.1 are also available.

```
model.LCIA.LCI.Components.Data.Area           % exchange area definition
model.LCIA.LCI.Components.Data.Pressure       % operating pressure definition
model.LCIA.LCI.Components.Units.Area          % units in which the area is defined
model.LCIA.LCI.Components.Units.Pressure      % units in which the pressure is defined
```

Listing D.11: HeatEx_ST

Field	Var. Type	Unit	Description
Data.Area	float	m ²	exchange area definition
Data.Pressure	float	bar	operating pressure definition
Units.Area	cell string		units in which the area is given
Units.Pressure	cell string		units in which the pressure is given

Table D.12: Generic shell and tube heat exchanger fields from listing D.11

D.7.8 Molecular_Sieve

Listing D.12 and table D.13 report fields that are used to define a molecular sieve. The life cycle inventory and the impact of the molecular sieve are scaled with respect to the diameter, the height and the pressure. Fields defined at sections D.4.1 and D.7.1 are also available.

```
model.LCIA.LCI.Components.Data.Diameter       % diameter definition
model.LCIA.LCI.Components.Data.Height         % height definition
model.LCIA.LCI.Components.Data.Pressure       % pressure definition
model.LCIA.LCI.Components.Units.Diameter      % units in which the diameter is defined
model.LCIA.LCI.Components.Units.Height       % units in which the height is defined
model.LCIA.LCI.Components.Units.Pressure      % units in which the pressure is defined
```

Listing D.12: Molecular_Sieve

Field	Var. Type	Unit	Description
Data.Diameter	float	m	diameter definition
Data.Height	float	m	height definition
Data.Pressure	float	bar	pressure definition
Units.Diameter	cell string		units in which the diameter is given
Units.Height	cell string		units in which the height is given
Units.Pressure	cell string		units in which the pressure is given

Table D.13: Generic molecular sieve fields from listing D.12

D.7.9 Pump

Listing D.13 and table D.14 report fields that are used to define a pump. The life cycle inventory and the impact of the pump are scaled with respect to the electrical power and the operating pressure. Fields defined at sections D.4.1 and D.7.1 are also available.

```

model.LCIA.LCI.Components.Data.Power      % electrical power definition
model.LCIA.LCI.Components.Data.Pressure    % operating pressure definition
model.LCIA.LCI.Components.Units.Power      % units in which the electrical power is
defined
model.LCIA.LCI.Components.Units.Pressure    % units in which the operating pressure is
defined

```

Listing D.13: Pump

Field	Var. Type	Unit	Description
Data.Power	float	kW	electrical power definition
Data.Pressure	float	bar	operating pressure definition
Units.Power	cell string		units in which the electrical power is given
Units.Pressure	cell string		units in which the operating pressure is given

Table D.14: Generic pump fields from listing D.13

D.7.10 Reactor_SS

Listing D.14 and table D.15 report fields that are used to define a reactor in stainless steel. The life cycle inventory and the impact are scaled with respect to the diameter, the height and the operating pressure. Fields defined at section D.4.1 and D.7.1 are also available.

```

model.LCIA.LCI.Components.Data.Diameter    % diameter definition
model.LCIA.LCI.Components.Data.Height      % height definition
model.LCIA.LCI.Components.Data.Pressure    % operating pressure definition
model.LCIA.LCI.Components.Units.Diameter    % units in which the diameter is defined
model.LCIA.LCI.Components.Units.Height      % units in which the height is defined
model.LCIA.LCI.Components.Units.Pressure    % units in which the pressure is defined

```

Listing D.14: Reactor_SS

Field	Var. Type	Unit	Description
Data.Diameter	float	m	diameter definition
Data.Height	float	m	height definition
Data.Pressure	float	bar	operating pressure definition
Units.Diameter	cell string		units in which the diameter is defined
Units.Height	cell string		units in which the height is defined
Units.Pressure	cell string		units in which the pressure is defined

Table D.15: Generic reactor in stainless steel fields from listing D.14

D.7.11 Tank_Concrete

Listing D.15 and table D.16 report fields that are used to define a tank. The life cycle inventory and the impact of the tank are scaled with respect to the volume and the height. Fields defined at sections D.4.1 and D.7.1 are also available.


```

model.LCIA.LCI.Components.Data.Volume % volume definition
model.LCIA.LCI.Components.Data.Height % height definition
model.LCIA.LCI.Components.Units.Volume % units in which the volume is defined
model.LCIA.LCI.Components.Units.Height % units in which the height is defined

```

Listing D.15: Tank_Concrete

Field	Var. Type	Unit	Description
Data.Volume	float	m3	volume definition
Data.Height	float	m	height definition
Units.Volume	cell string		units in which the volume is given
Units.Height	cell string		units in which the height is given

Table D.16: Generic tank concrete fields from listing D.15

D.7.12 Turbine

Listing D.16 and table D.17 report fields that are used to define a turbine. The life cycle inventory and the impact are scaled with respect to the electrical power. Fields defined at sections D.4.1 and D.7.1 are also available.

```

model.LCIA.LCI.Components.Data.Power % electrical power definition
model.LCIA.LCI.Components.Units.Power % units in which the electrical power is defined

```

Listing D.16: Turbine

Field	Var. Type	Unit	Description
Data.Power	float	kW	electrical power definition
Units.Power	cell string		units in which the electrical power is defined

Table D.17: Generic turbine fields from listing D.16

D.8 User-defined parameters

Some parameters to perform a LCIA in energy technology have to be defined by the user in the frontend. These parameters are contained in the `o.LCIA` structure. Listing D.17 and table D.18 report the first-level fields that have to or that can be defined by the user in the frontend. The details of the `o.LCIA.FunctionalUnit` structure are reported at section D.8.

```

o.LCIA.ImpactMethod          % Definition of the impact assessment methods to be used
o.LCIA.LifeCycleStageMode    % Allows to work in life cycle stage mode, and have at
    the end of the computation a full detailed calculation for the overall life cycle (
    recommended mainly for multi-period problems)
o.LCIA.IncludeProcessEquipment % Allows to include process equipment (components) in the
    LCIA
o.LCIA.FunctionalUnit        % Definition of the parameters of the functional unit
o.LCIA.Graphics              % Definition of the parameters to plot contributions for a one
    run
o.LCIA.Params                % Definition of the general parameters linked with the
    system life cycle

```

Listing D.17: Content of the `o.LCIA` structure in the frontend

Field	Var. Type	Unit	Definition
<code>o.LCIA.ImpactMethod</code>	<code>cell string</code>		Definition of the impact assessment method to be used. Possible impact assessment methods that can be used in Energy Technologies are presented at section D.8.1
<code>o.LCIA.LifeCycleStageMode</code>	<code>bool</code>		Allows to work in life cycle stage mode, 1 means that difference has to be made between the different life cycle stages, 0 means not
<code>o.LCIA.IncludeProcessEquipment</code>	<code>bool</code>		Allows to include components of process equipment in the LCIA, 1 means that process equipment is to be included, 0 means it is not

Table D.18: Generic fields from listing D.17

Functional Unit definition

The fields of the `o.LCIA.FunctionalUnit` structure are reported at listing D.18 and table D.19.

```

o.LCIA.FunctionalUnit.Value    % Default value of the quantity of FU involved in the
    system lifetime
o.LCIA.FunctionalUnit.Unit     % Units in which the FU is given
o.LCIA.FunctionalUnit.Description % Description of the FU

```

Listing D.18: Content of the `o.LCIA.FunctionalUnit` structure

Field	Var. Type	Unit	Definition
<code>o.LCIA.FunctionalUnit.Value</code>	<code>float</code>	-	Default value of the quantity of FU involved in the system lifetime
<code>o.LCIA.FunctionalUnit.Unit</code>	<code>cell string</code>		Units in which the FU is given
<code>o.LCIA.FunctionalUnit.Definition</code>	<code>cell string</code>		Description of the FU

Table D.19: Generic fields from listing D.18

Graphics parameters definition

The fields of the `o.LCIA.Graphics` structure are reported at listing D.19 and table D.20.

```

o.LCIA.Graphics.PlotContribution      % Allows to plot bar charts of impact contributions
o.LCIA.Graphics.PlotContributionGlobal % Allows to plot bar charts of overall impact
                                     contributions, in life cycle stage mode
o.LCIA.Graphics.PlotLifeCycleStage   % Allows to plot bar charts of the contributions of
                                     the three life cycle stages, in life cycle stage mode
o.LCIA.Graphics.PlotModelContribution % Allows to plot bar charts of the contributions
                                     of the different models
o.LCIA.Graphics.PlotModelContributionGlobal % Allows to plot bar charts of the
                                     contributions of the different models, in life cycle stage mode
o.LCIA.Graphics.CutOff               % Definition of cut-off applied for plotted impact
                                     contributions
o.LCIA.Graphics.SavePlot             % Allows to choose for saving the plots

```

Listing D.19: Content of the `o.LCIA.Graphics` structure

Field	Var. Type	Unit	Definition
<code>o.LCIA.Graphics.PlotContribution</code>	<code>bool</code>		Allows to plot bar charts of impact contributions, 1 means that plots are made, 0 means not
<code>o.LCIA.Graphics.PlotContributionGlobal</code>	<code>bool</code>		Allows to plot bar charts of overall impact contributions in life cycle stage mode, 1 means that plots are made, 0 means not
<code>o.LCIA.Graphics.PlotLifeCycleStage</code>	<code>bool</code>		Allows to plot bar charts of the contributions of the three life cycle stages, 1 means that plots are made, 0 means not
<code>o.LCIA.Graphics.PlotModelContribution</code>	<code>bool</code>		Allows to plot bar charts of the contributions of the different models, 1 means that plots are made, 0 means not
<code>o.LCIA.Graphics.PlotModelContributionGlobal</code>	<code>bool</code>		Allows to plot bar charts of the contributions of the different models in life cycle stage mode, 1 means that plots are made, 0 means not
<code>o.LCIA.Graphics.CutOff</code>	<code>float</code>	-	Definition of cut-off applied for plotted impact contributions, relative contributions inferiores to this value will be all included in a single Remaining Processes contribution
<code>o.LCIA.Graphics.SavePlot</code>	<code>bool</code>		Allows to choose for saving the plots, 1 means they are saved, 0 means not

Table D.20: Generic fields from listing D.19

General Life Cycle Parameters

The fields of the `o.LCIA.Params` structure are reported at listing D.20 and table D.21.

```

o.LCIA.Params.Lifetime % Lifetime of the system, applied for each component if no other
specification
o.LCIA.Params.MaintenanceTimeRatio % Yearly percentage of total impact due to maintenance
, applied for each component if no other specification
o.LCIA.Params.TransportDistance % Transport distance for life cycle after production phase
, applied for each component if no other specification
o.LCIA.Params.RecyclingRatio % Percentage of recycling at end-of-life, leaving
system limits, applied for each component if no other specification
o.LCIA.Params.TransportID % Equivalence for life cycle transport
o.LCIA.Params.DisposalID % Equivalence for disposal (part not recycled)

```

Listing D.20: Content of the `o.LCIA.Params` structure

Field	Var. Type	Unit	Definition
<code>o.LCIA.Params.Lifetime</code>	<code>float</code>	years	Lifetime of the system, applied for each component if no other specification
<code>o.LCIA.Params.MaintenanceTimeRatio</code>	<code>float</code>	-	Yearly percentage of total impact due to maintenance, applied for each component if no other specification
<code>o.LCIA.Params.TransportDistance</code>	<code>float</code>	km	Transport distance for life cycle after production phase, applied for each component if no other specification
<code>o.LCIA.Params.RecyclingRatio</code>	<code>float</code>	-	Percentage of recycling at end-of-life, leaving system limits, applied for each component if no other specification
<code>o.LCIA.Params.TransportID</code>	<code>cell string</code>		Equivalence for life cycle transport, unit process ID
<code>o.LCIA.Params.DisposalID</code>	<code>cell string</code>		Equivalence for disposal (part not recycled), process ID

Table D.21: Generic fields from listing [D.20](#)

D.8.1 Impact assessment methods

The different impact assessment methods, taken from ecoinvent, that can be handled in Energy Technologies are listed at table [D.22](#). A short description of each impact assessment method is provided at the following subsections. For more information, (donner une reference d'un rapport ecoinvent).

Short Name	Complete Name
<code>ced</code>	Cumulative Energy Demand
<code>cml01_ext</code>	CML 2001, all categories
<code>cml01_short</code>	CML 2001, only four categories
<code>ei99ee</code>	Ecoindicator99, egalitarian perspective
<code>ei99ha</code>	Ecoindicator99, hierarchist perspective
<code>ei99ii</code>	Ecoindicator99, individualist perspective
<code>footprint</code>	Ecological footprint
<code>impact02_end</code>	Impact2002+, endpoint categories
<code>ipcc01</code>	International Panel on Climate Change, 2001
<code>ipcc07</code>	International Panel on Climate Change, 2007
<code>ubp</code>	ecological scarcity 2006 (UmweltBelastungsPunkte)

Table D.22: Listing of impact assessment methods that are implemented in Energy Technologies

Cumulative Energy Demand

This method is used to calculate the energy use throughout the life cycle of a good or a service. It includes direct and indirect (or grey) energy involved. The impact categories available are reported at table [D.23](#).

Category	Unit
Non renewable, fossil	MJ-eq
Non renewable, nuclear	MJ-eq
Non renewable, primary forest	MJ-eq
Renewable, biomass	MJ-eq
Renewable, wind	MJ-eq
Renewable, solar	MJ-eq
Renewable, geothermal	MJ-eq
Renewable, water	MJ-eq

Table D.23: Listing of impact categories for `ced`

CML 2001

This method uses a problem-oriented approach (by opposition to a damage-oriented approach, like Ecoindicator99) for and aggregates the elements of the substance vectors in equivalent of different substances representing impact categories. Categories are also divided in different time ranges for which the problem is considered. The impact categories of the `cm101_ext` are reported at listing [D.24](#). For `cm101_short`, only four categories which are some of the most currently used are included, and reported at listing [D.25](#).

Category	Unit
acidification potential, average European	kg SO2-eq
acidification potential, generic	kg SO2-eq
climate change, GWP 100a	kg CO2-eq
climate change, GWP 20a	kg CO2-eq
climate change, GWP 500a	kg CO2-eq
climate change, lower limit of net GWP	kg CO2-eq
climate change, upper limit of net GWP	kg CO2-eq
eutrophication potential, average European	kg NOx-eq
eutrophication potential, generic	kg PO4-eq
freshwater aquatic ecotoxicity, FAETP 100a	kg 1,4-DCB-eq
freshwater aquatic ecotoxicity, FAETP 20a	kg 1,4-DCB-eq
freshwater aquatic ecotoxicity, FAETP 500a	kg 1,4-DCB-eq
freshwater aquatic ecotoxicity, FAETP infinite	kg 1,4-DCB-eq
human toxicity, HTP 100a	kg 1,4-DCB-eq
human toxicity, HTP 20a	kg 1,4-DCB-eq
human toxicity, HTP 500a	kg 1,4-DCB-eq
human toxicity, HTP infinite	kg 1,4-DCB-eq
land use, competition	m2a
marine aquatic ecotoxicity, MAETP 100a	kg 1,4-DCB-eq
marine aquatic ecotoxicity, MAETP 20a	kg 1,4-DCB-eq
marine aquatic ecotoxicity, MAETP 500a	kg 1,4-DCB-eq
marine aquatic ecotoxicity, MAETP infinite	kg 1,4-DCB-eq
photochemical oxidation (summer smog), EBIR	kg formed O3
photochemical oxidation (summer smog), MIR	kg formed O3
photochemical oxidation (summer smog), high NOx POCP	kg formed O3
photochemical oxidation (summer smog), low NOx POCP	kg formed O3
resources, depletion of abiotic resources	kg Sb-eq
stratospheric ozone depletion, ODP 10a	kg CFC-11-eq
stratospheric ozone depletion, ODP 15a	kg CFC-11-eq
stratospheric ozone depletion, ODP 20a	kg CFC-11-eq
stratospheric ozone depletion, ODP 25a	kg CFC-11-eq
stratospheric ozone depletion, ODP 30a	kg CFC-11-eq
stratospheric ozone depletion, ODP 40a	kg CFC-11-eq
stratospheric ozone depletion, ODP 5a	kg CFC-11-eq
stratospheric ozone depletion, ODP steady state	kg CFC-11-eq
terrestrial ecotoxicity, TAETP 100a	kg 1,4-DCB-eq
terrestrial ecotoxicity, TAETP 20a	kg 1,4-DCB-eq
terrestrial ecotoxicity, TAETP 500a	kg 1,4-DCB-eq
terrestrial ecotoxicity, TAETP infinite	kg 1,4-DCB-eq
freshwater sediment ecotoxicity, FSETP 100a	kg 1,4-DCB-eq
freshwater sediment ecotoxicity, FSETP 20a	kg 1,4-DCB-eq
freshwater sediment ecotoxicity, FSETP 500a	kg 1,4-DCB-eq
freshwater sediment ecotoxicity, FSETP infinite	kg 1,4-DCB-eq
malodours air	m3 air
marine sediment ecotoxicity, MSETP 100a	kg 1,4-DCB-eq
marine sediment ecotoxicity, MSETP 20a	kg 1,4-DCB-eq
marine sediment ecotoxicity, MSETP 500a	kg 1,4-DCB-eq
marine sediment ecotoxicity, MSETP infinite	kg 1,4-DCB-eq
ionising radiation	DALYs

Table D.24: Listing of impact categories implemented in `cm101_ext`

Category	Unit
acidification potential, average European	kg SO ₂ -eq
climate change, GWP 100a	kg CO ₂ -eq
eutrophication potential, generic	kg PO ₄ -eq
stratospheric ozone depletion, ODP steady state	kg CFC-11-eq

Table D.25: Listing of impact categories implemented in `cm101_short`

Ecoindicator99

This method uses a damage-oriented approach (by opposition to a problem-oriented approach, like CML2001). Damages are finally normalized and weighted, to be expressed in ecopoints. Three different weighting sets are available, representing three different perspectives: the hierarchist perspective, the egalitarian perspective, and the individualist perspective. Though these weightings are subjective, it is agreed that the hierarchist perspective represent the scientific point of view. The impact categories available are reported at table D.26. The category of the single score is the total sum of the three above impact categories.

Category	Unit
Human Health	pts (DALYs)
Ecosystem Quality	pts (PDF)
Resources	pts (MJ-surplus)
Single Score	pts (-)

Table D.26: Listing of impact categories implemented in `ei99ee`, `ei99ha` and `ei99ii`

Ecological Footprint

This method uses the concept of ecological footprint as an impact assessment method. The ecological footprint refers as the biologically productive land and water a population requires to produce the resources it consumes and to absorb the part of the waste generated by fossil and nuclear fuel consumption.

Category	Unit
CO ₂	m ² a
Land occupation	m ² a
Total	m ² a
Nuclear	m ² a

Table D.27: Listing of impact categories implemented in `footprint`

Impact 2002+

This method proposes a combined midpoint (or problem) and damage approach, though only damage approach is available in Energy Technologies. The impact categories available are reported at table D.28. The impact categories available are reported at table D.27.

Category	Unit
Human Health	pts (DALYs)
Ecosystem Quality	pts (PDF)
Climate Change	pts (kgCO ₂ -eq)
Resource	pts (MJ-eq)

Table D.28: Listing of impact categories implemented in `impact02_end`

IPCC - climate change

This method uses the characterisation factors of different gaseous emissions to calculate the global warming potential. The characterisation factors are based on the global warming potentials published by the IPCC (International Panel on Climate Change). The impact categories available are reported at table [D.29](#).

Category	Unit
GWP 20a	kgCO ₂ -eq
GWP 100a	kgCO ₂ -eq
GWP 500a	kgCO ₂ -eq

Table D.29: Listing of impact categories implemented in [ipcc01](#) and [ipcc07](#)

Ecological Scarcity

This method, also called 'Umweltbelastungspunkte' uses characterization factors for different types of emissions, energy resources and waste, that aggregates them to a present pollution level, and compare them with a critical pollution level, that is based on the scientifically supported goals of the Swiss environmental policy. It is therefore recommended that this method is used only in a study having a Swiss context. The impact categories available are reported at table [D.30](#). The category of the single score is the total sum of the seven above impact categories.

Category	Unit
Emissions into Air	UBP
Emissions into Surface Water	UBP
Emissions into Ground-Water	UBP
Emissions into Top Soil	UBP
Energy Resources	UBP
Natural Resources	UBP
Deposited Waste	UBP
Single Score	UBP

Table D.30: Listing of impact categories implemented in [ubp](#)

D.9 ecoinvent database

The ecoinvent database has to be downloaded from svn to use the LCA module. It is available at: <https://leni.epfl.ch/svn/LCIA/ecoinvent>.

If any update of the database is made, it is necessary to delete first the folder named 'LCIAMat' in the model folder to make the update effective!

D.10 Quicktips

D.10.1 Pre or Post LCIA function

If some calculations have to be made right before or right after the LCIA, for example to update the LCI after the energy integration or to post-treat the results after the LCA, a pre- or post- LCIA function can be written and declared in the model description file. The syntax is explained at listing [D.21](#) and table [D.31](#).


```

model.PreLCIAMFunction % function(s) to be called before the LCIA calculations
model.PostLCIAMFunction % function(s) to be called after the LCIA calculations

```

Listing D.21: Pre and Post LCIA functions

Field	Var. type	Unit	Description
PreLCIAMFunction	cell string		name of the matlab function(s) to be run before the LCIA calculations
PostLCIAMFunction	cell string		name of the matlab function(s) to be run after the LCIA calculations

Table D.31: Declaration of Pre- and Post- LCIA functions from listing D.21

An example of this syntax is given at listing D.22.

```

model.PreLCIAMFunction = {'PreLCIA_function1','PreLCIA_function2'}

```

Listing D.22: Example of syntax for Pre or Post LCIA functions

D.10.2 Functions to update LCI data

In a pre-LCIA function, it can be useful to update the LCI data. Three functions can be used for this, to update either the unit processes, the elementary flows or the components. These are detailed at listing D.23.

```

o = LCIA_update_components_tags(o, Components) % function to update the components, which
new information is contained in the structure 'Components'
o = LCIA_update_eflows_tags(o, EFlows) % function to update the elementary flows, which
new information is contained in the structure 'EFlows'
o = LCIA_update_uprocess_tags(o, UProcess) % function to update the unit processes, which
new information is contained in the structure 'UProcess'

```

Listing D.23: Functions to be used for updating LCI data

An example of this syntax is given at listing D.24.

```

EFlows = struct;
i=0;
i=i+1;
EFlows(i).TagName = {'CO2_emitted'};
EFlows(i).Value = emission_rate_co2 * net_power;
i=i+1;
EFlows(i).TagName = {'NOx_emitted'};
EFlows(i).Value = emission_rate_nox * net_power;
o = LCIA_update_eflows_tags(o,EFlows);

```

Listing D.24: Functions to be used for updating LCI data

Warning: for updating the components, it is necessary to update simultaneously all the data for the components to be updated!

D.11 Contact

If any problem related to the use of the LCA module is encountered, the person to be contacted is: leda.gerber@epfl.ch .

Appendix E

Subversion Information

These information are useful for svn debugging purposes. Most of these information are relative to this sub-file of the main documentation project.

This file: svn_infos.tex
Id of this file: svn_infos.tex 335 2008-08-11 15:01:43Z bolliger
Current Year: 2008
Revision of this file: 335
Current Month: 8
Current Day: 11
Minimum Documentation Project Revision: 302
Maximum Documentation Project Revision: 2455
Last Revision Date: August 11, 2008
Short Date: 2008-08-11
Today: August 11, 2008
Time: 15:01:43Z
Author of this file: bolliger