Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Départmenet fédéral de l'environnement, des transports,
de l'énergie et de la communication DETEC

**Ofiice fédéral de l'énergie OFEN**

**Rapport final / Rapport annuel** juin 2009

# EasyPipes

**Mandant:**
Office fédéral de l'énergie  OFEN
Programme de recherche Energie dans les bâtiments
CH-3003  Berne
www.bfe.admin.ch

**Cofinancement:**
Institution AB, CH-6666  Lieu
Institution BC, CH-7777  Lieu

**Mandataire:**
CUEPE
Institut des Sciences de l'Environnement de l'Université de Genève
Université de Genève
7, Route de Drize, CH-1227 Carouge, Genève
www.cuepe.ch

**Auteurs:**
Peter Gallinelli, Université de Genève, peter.gallinelli@leea.ch
Pierre Hollmuller, Université de Lisbonne, pierre.hollmuller@fc.ul.pt
Pascal Thomann, Université de Genève, pascal.thomann@leea.ch
Willi Weber, Université de Genève, willi.weber@unige.ch

**Responsable de domaine de l'OFEN:** Andreas Eckmanns
**Chef de programme de l'OFEN:** Charles Filleux
**Numéro du contrat et du projet de l'OFEN:** 153000 / 102000

# Introduction

## Contexte et état de l'art

Parmi les concepts alternatifs pour le chauffage et le rafraîchissement de bâtiments à faible consommation d'énergie, il est de plus en plus fréquemment fait usage d'échangeurs air/sol (puits canadiens), sans pour autant que les outils de dimensionnement appropriés soient à disposition.

En effet, on rencontre dans la littérature toute une série d'algorithmes de calcul pour échangeurs air/sol, analytiques ou par éléments finis, avec des niveaux de finesse et complexités très variés. Cependant, la plupart de ces outils n'ont pas fait l'objet de validations extensives et ne sont généralement pas disponibles sous forme opérationnelle.

Dans ce contexte et déjà sous mandat de l'Office fédéral de l'énergie, le CUEPE a précédemment développé un algorithme de calcul par éléments finis (projet OFEN no 17'507), qui:

- permet une description flexible de la géométrie (sols inhomogènes, plusieurs nappes de tubes), prend en compte diverses conditions au bord (notamment le couplage avec le bâtiment), intègre les échanges latents entre air et tube (évaporation et condensation), la diffusion de chaleur en trois dimensions et les pertes de charge le long des tubes,
- a fait l'objet de validations extensives, avec d'une part une solution analytique complète en géométrie simple (symétrie cylindrique, flux constant), d'autre part un ensemble de mesures longues durées sur des systèmes réels (y compris échanges latents),
- est intégré dans une routine TRNSYS (Type 460), mise à disposition de la communauté scientifique et des grands bureaux d'ingénieurs, permettant son intégration modulaire dans la simulation dynamique de systèmes énergétiques.

Son utilisation généralisée souffre cependant du passage obligé par un fichier de paramétrisation, contenant le détail de la géométrie, dont l'édition manuelle reste complexe.

## Objectifs et groupe cible

Fort des instruments de simulation et d'analyse précédemment acquis, ce projet propose de développer un outil de dimensionnement pour échangeurs air/sol facile à l'emploi. Ce développement, qui se fera en deux temps, donnera lieu aux deux produits spécifiques suivants :

- La mise à niveau de la routine TRNSYS (Type 460), avec génération automatique du fichier de paramétrisation pour les géométries simples (tube unique ou nappe de tubes unique, sol homogène), qui représentent la grande majorité des cas étudiés,
- Le développement d'une interface utilisateur graphique (GUI) qui intègre la routine TRNSYS dans un environnement dédié, permettant au groupe d'utilisateurs cible de s'affranchir de l'environnement TRNSYS et d'avoir accès à un outil de simulation et d'analyse intégré.

L'un et l'autre de ces produits correspondent à une demande répétée des milieux concernés: bureaux d'étude et architectes spécialisés confrontés au dimensionnement d'échangeurs air/sol, ainsi que laboratoires de recherche en thermique du bâtiment et stockage de chaleur.

# Mise à niveau de la routine TRNSYS

La mise à jour de la routine TRNSYS (Type 460) avait pour objectif à la fois de simplifier son usage dans le cadre de l'environnement TRNSYS, et de permettre son utilisation avec l'interface Easypipes.

A cet effet, les modifications suivantes ont été apportées à la routine FORTRAN (qui compte env. 3800 lignes de code).

## Géométrie : mode simplifié versus mode expert

Pour le cas de loin le plus fréquent d'une géométrie simple (sol homogène, tubes alignés, condition de surface unique), et afin d'éviter l'édition fastidieuse du fichier paramètre, l'activation d'un flag permet dorénavant de générer automatiquement la géométrie (notamment le maillage), à partir des paramètres de base suivants :

- TypAir : type de flux d'air (volumique ou massique)
- NtubY : nombre de tubes en parallèle (axe y)
- NtubZ : nombre de tubes superposés (axe z)
- LsoilY : entre-axe entre tubes parallèles
- LsoilZ : entre-axe entre tubes superposés
- LsoilTop : épaisseur de sol superficielle (au dessus du premier tube)
- LsoilBot : épaisseur de sol inférieure (au dessous du dernier tube)
- LsoilSide : épaisseur de sol latérale (au dessous du dernier tube)
- Ltub :longueur des tubes
- Dtub : diamètre des tubes
- ThTub : épaisseur des tubes
- LamTub : conduction thermique tubes
- CvTub : capacité thermique tubes
- CtubFric : coefficient de frictiontubes
- LamSoil : conduction thermique sol
- CvSoil : capacité thermique sol
- TiniSoil : température initiale sol
- TypSurfTop : type de surface supérieure (input en température/puissance)
- RsurfTop : résistance de surface supérieure
- TypSurfTop : type de surface inférieure (input en température/puissance)
- RsurfTop : résistance de surface inférieure

La géométrie ainsi générée est transcrite dans un fichier de contrôle (qui peut le cas échéant servir de base au fichier paramètre requis par le mode expert).

En alternative au mode simplifié, l'activation du mode expert (avec lecture de fichier paramètre) reste disponible, notamment pour le cas de géométries plus complexes (sols inhomogènes, tubes en quinconce, conditions de surface multiples).

## Input / Output

Outre la mise en place du mode de lecture simplifié décrit ci-dessus, la structure d'appel de la routine a été modifiée de la façon suivante :

- Les unités physiques des variables ont été ramenées dans le système standard MKSA, à l'exception des cas particulier suivant : temps (h) et débits (kg/h ou m3/h).
- En mode expert, par souci de cohérence, les paramètres à passer à la routine ont été intégralement regroupés dans le fichier correspondant.
- Le nombre d'input/output est automatiquement adapté au nombre de surfaces données par la définition géométrique, variable en mode expert, fixe en mode simplifié (une surface supérieure, une surface inférieure).

## Améliorations diverses

Parmi les diverses améliorations apportées, on nommera en particulier :

- La possibilité de choisir entre flux d'air volumétrique (m3/h) ou massique (kg/h).
- L'implémentation d'une température output pour le cas d'un débit d'air nul (température d'air égalée à la température de tube en sortie du puits canadien).
- Le calcul automatique de l'échange convectif air/tube (formulation de Gnielinski), en alternative à sa définition phénoménologique (forme linéarisée en fonction de la vitesse), qui reste cependant possible en mode expert.
- La vérification de saisie et les messages d'erreurs associés ont été améliorés.

## Bugs

Les bugs connus ont été corrigés :

- Calcul de la résistance de couche superficielle (Kbord) tenant correctement compte de la résistance de surface (Rsurf).
- Calcul automatique du débit minimum (XairMin).
- Calcul des outputs optionnels de vitesse et débit d'air (Vair et Xair)

# Interface EasyPipes

L’interface graphique du logiciel se présente sous la forme d’un bureau avec une barre de menus et d’outils. Une fenêtre de saisie, une console de dialogue logiciel et une fenêtre de résultats s’affichent sur la zone de ‘bureau’.

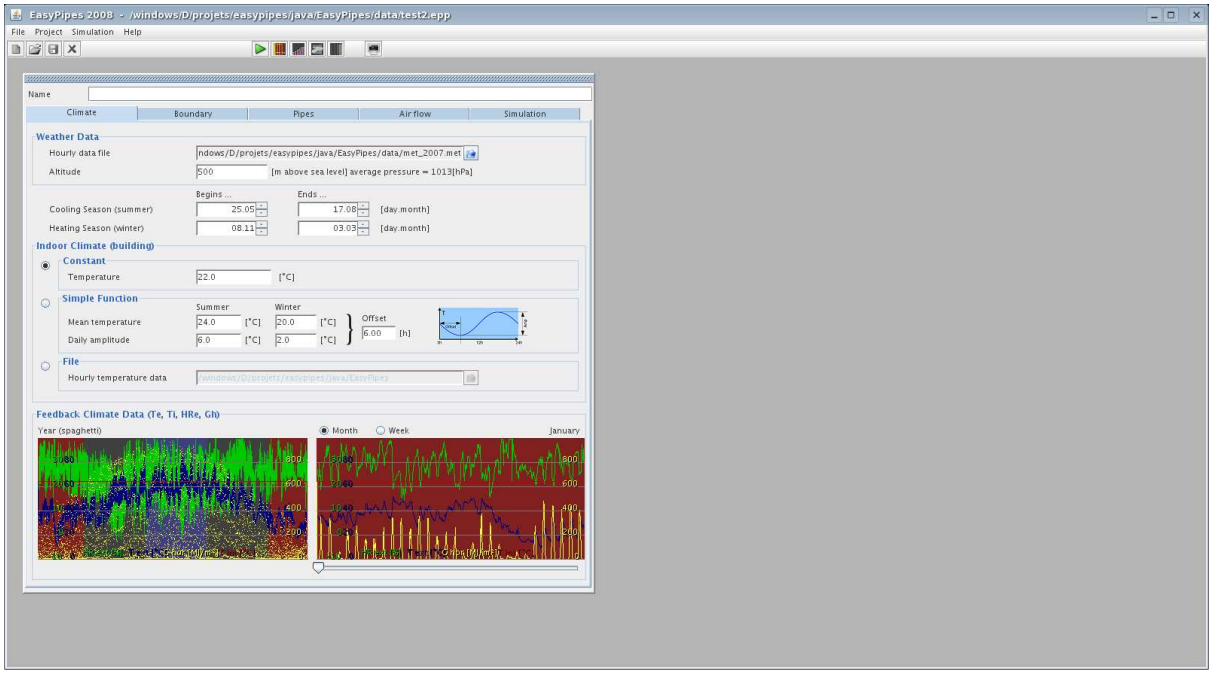


Figure 1 : le bureau EasyPipes

L’utilisateur devra définir un modèle de simulation en renseignant successivement les onglets de saisie. La simulation qui dure plusieurs minutes est déclenchée par l’utilisateur. Au terme des calculs, les résultats peuvent être affichés sous forme de graphiques synthétiques annuels, mensuels et hebdomadaires.

S’adressant à un public international, l’anglais a été retenu comme langue d’interface.

## Fenêtre de saisie (input)

L'interface de offre cinq onglets structurés par thèmes, présentés dans l’ordre de saisie et d’exécution du projet de simulation :

- Climate : données climatiques extérieures et intérieures
- Boundaries : conditions de surface et prise d’air
- Pipes : caractéristiques et géométrie des tubes enterrés
- Air flow : débits de ventilation et caractéristiques de la régulation
- Simulation : paramètres de simulation, ‘RUN’ (exécution de la simulation) et notes de projet

**Onglet ‘Climate’**

Les données climatiques proviennent d’un fichier horaire au format ‘CSV’[1].

L’altitude est requise pour déterminer les caractéristiques moyennes de l’atmosphère selon le modèle d’atmosphère standard U.S. 1976.

Trois saisons sont définies par des ‘spinners’ :

– Saison de refroidissement (cool)

– Saison de chauffage (heat)

– Saison intermédiaire (mids)

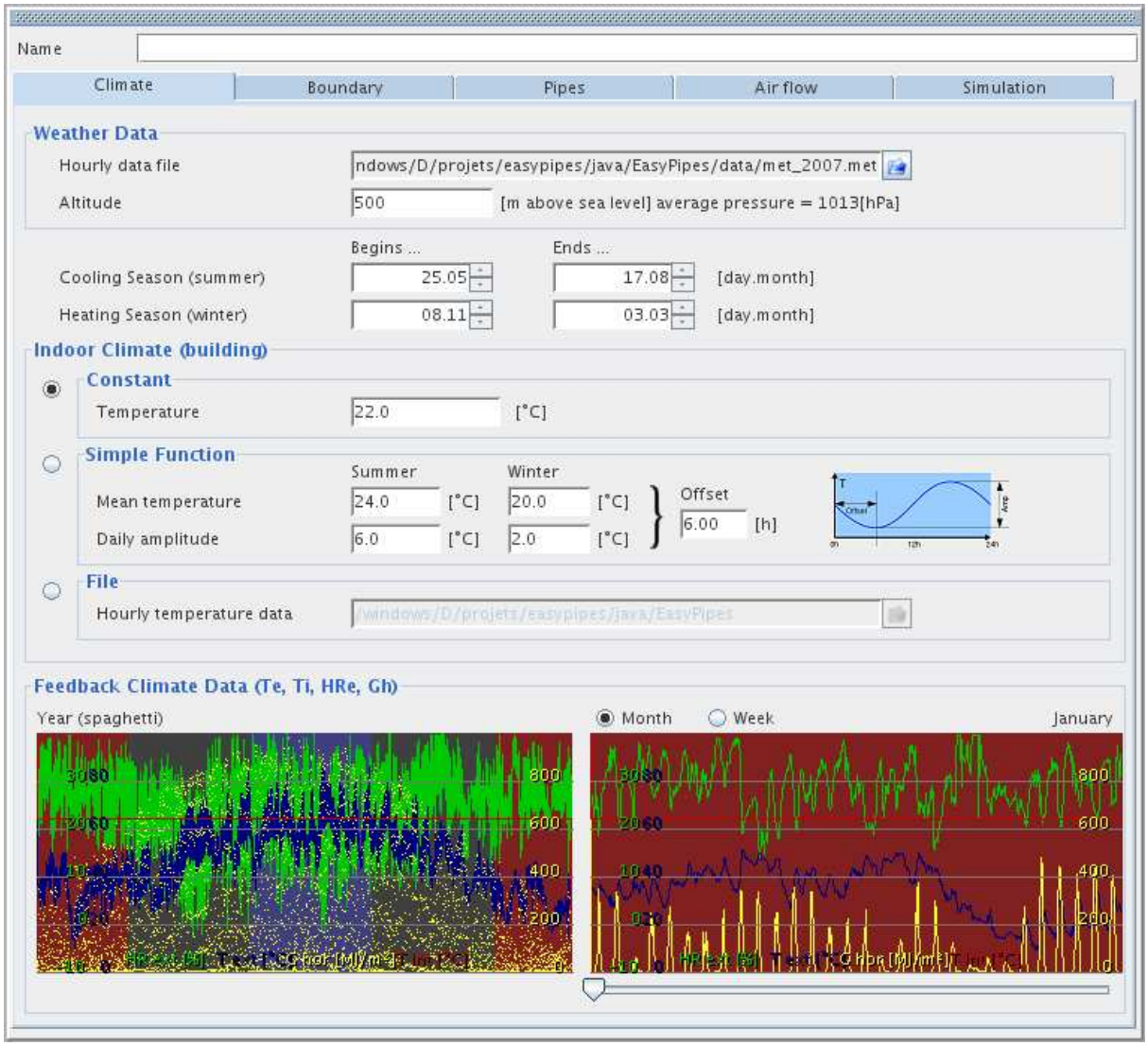


Figure 2 : capture onglet ‘Climate’

Le climat intérieur (température) peut être défini au choix de trois manières :

– Température constante,

– Fonction simple (sinusoïdale été, hiver avec une période de transition par interpolation linéaire),

[1] Comma-separated values (CSV) est un format informatique ouvert représentant des données tabulaires sous forme de « valeurs séparées par des virgules ». Un fichier CSV est un fichier texte (par opposition aux formats dit « binaires »). Chaque ligne correspond à une rangée du tableau et les cellules d'une même rangée sont séparées par une virgule.

– Profil horaire défini par l'utilisation dans un fichier horaire au format 'CSV'.

Un retour d'information par affichage direct des données de température, humidité et rayonnement sur l'année et par zoom mensuel et hebdomadaire permet un contrôle instantané des données d'entrée.

**Onglet 'Boundary'**

L'air à l'entrée des tubes peut provenir au choix :

– Des données climatiques défini au chapitre 'Climate'

– D'un fichier horaire au format 'CSV'

– Trois configurations de disposition des tubes par rapport au bâtiment sont possibles :

– Tubes en dehors du périmètre du bâtiment. Dans ce cas sont pris en compte la résistance d'échange thermique superficiel et l'absorptivité du terrain, nécessaires pour évaluer une température de surface équivalente,

– Tubes sous le bâtiment avec transfert de chaleur entre le bâtiment et le sol,

– Tubes enterrés sans transfert de chaleur avec la surface (adiabatique).

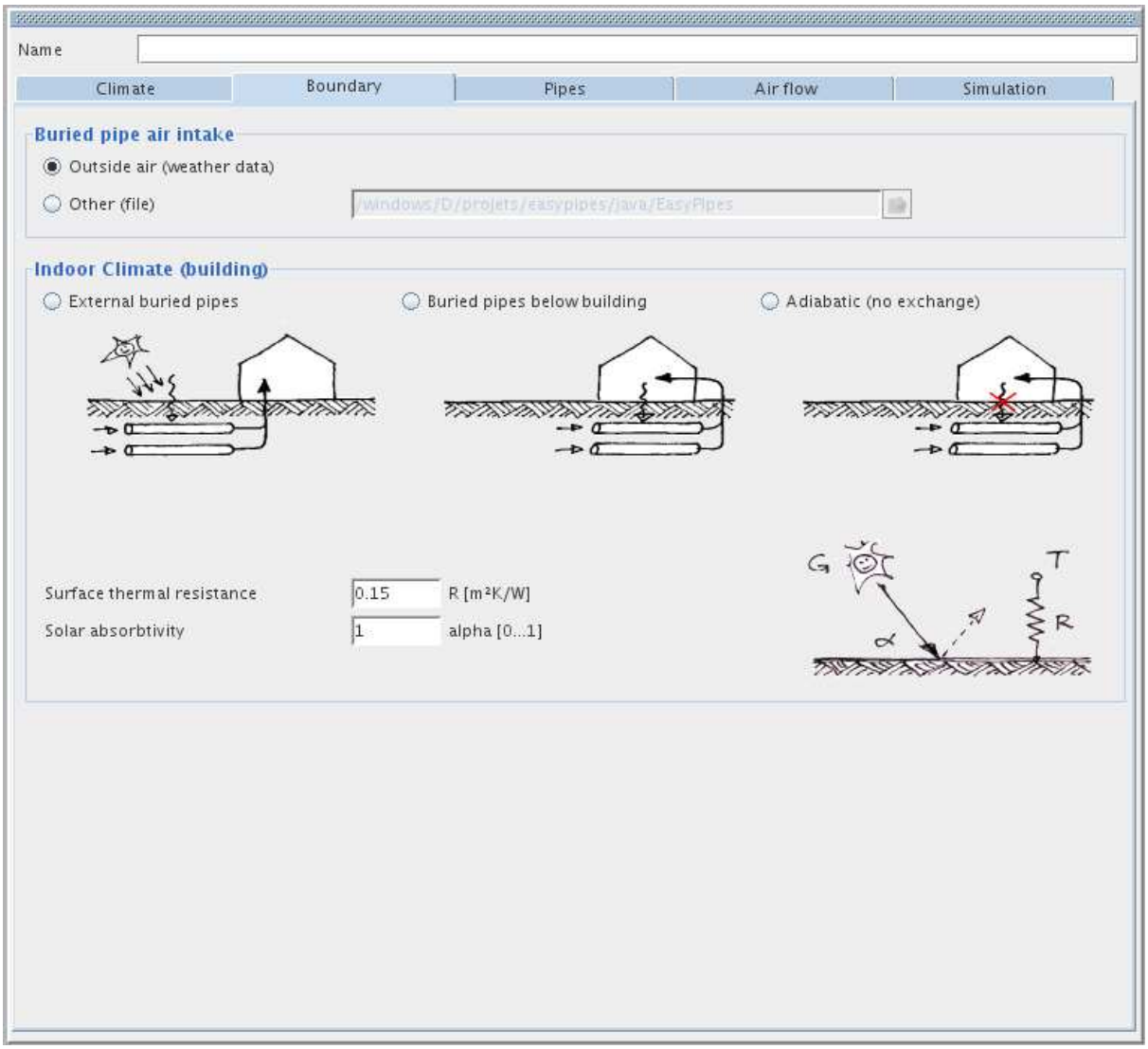


Figure 3 : capture onglet 'Boudary'

**Onglet ‘Pipes’**

L’onglet ‘Pipes’ définit :

– Les propriétés du terrain,

– Les propriétés physiques et dimensions des tubes,

– La géométrie du réseau de tubes

Un feed-back par 4 chiffres caractéristiques permet un contrôle de saisie.

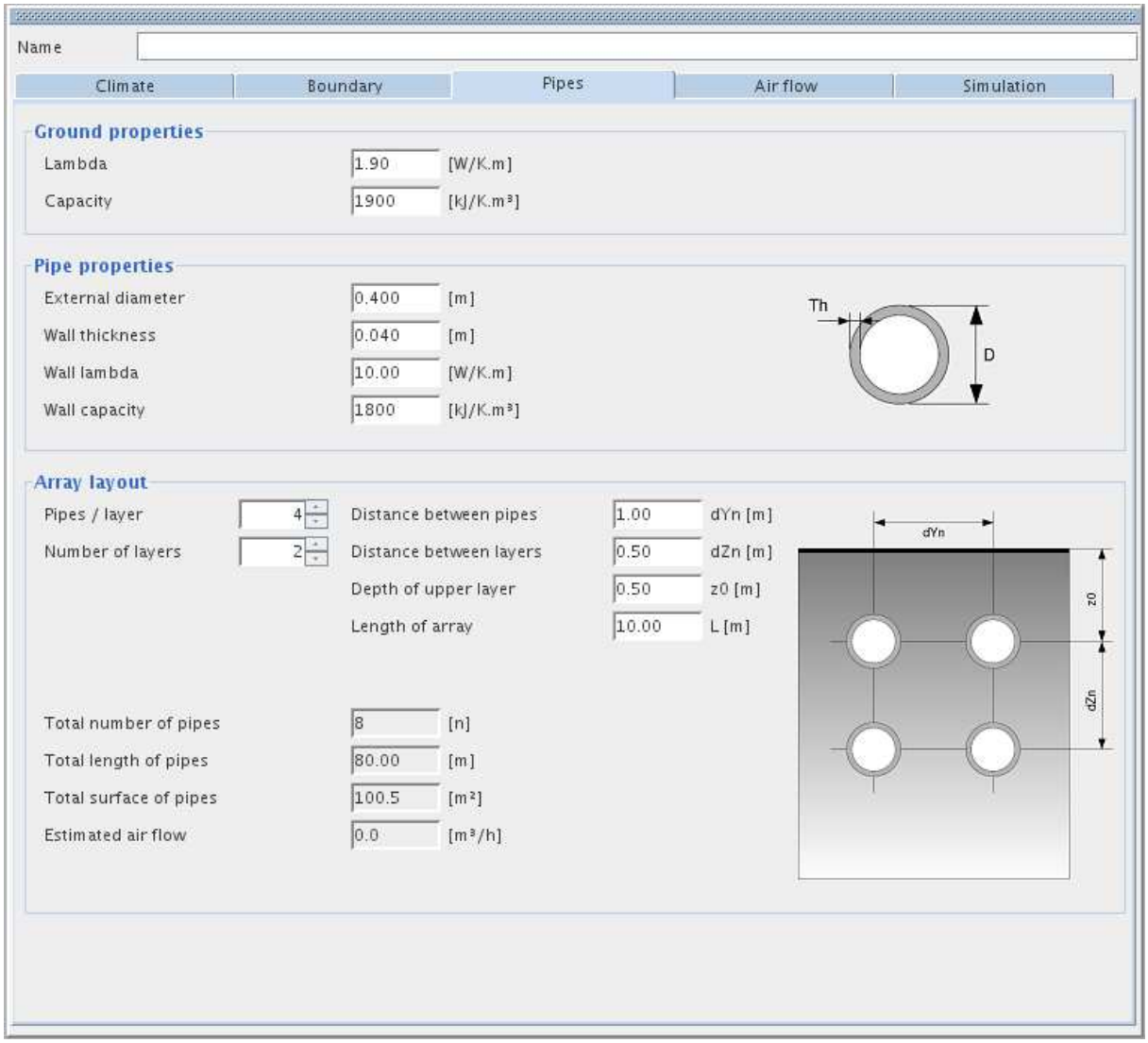


Figure 4 : capture onglet ‘Pipes’

**Onglet ‘Air flow’**

L’onglet ‘Air flow’ définit les caractéristiques de ventilation. Le débit de ventilation peut être fixé de trois manières :

– Débit constant (m³/h),

– Horaire librement configurable (listes d’heures et de débits (m³/h),

– Par régulation (voir ci-dessous).

Un feed-back graphique offre un contrôle instantané des saisies.


Figure 5 : capture onglet ‘Air flow’

**Régulation de la ventilation**

La régulation tient compte de deux ou trois températures données par des sondes ou par des préréglages (setpoints). Pour chaque cas, deux débits sont possibles :

– Condition remplie : débit-true (m³/h)

– Sinon : débit-false (m²/h)

Ci-dessous les huit configurations proposées par EasyPipes :

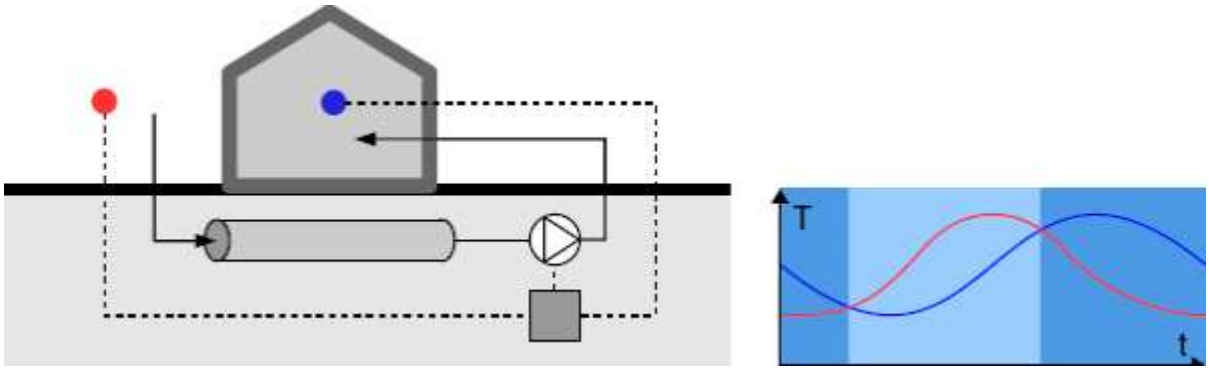


Figure 6 : régulation sonde T ext. - T int.

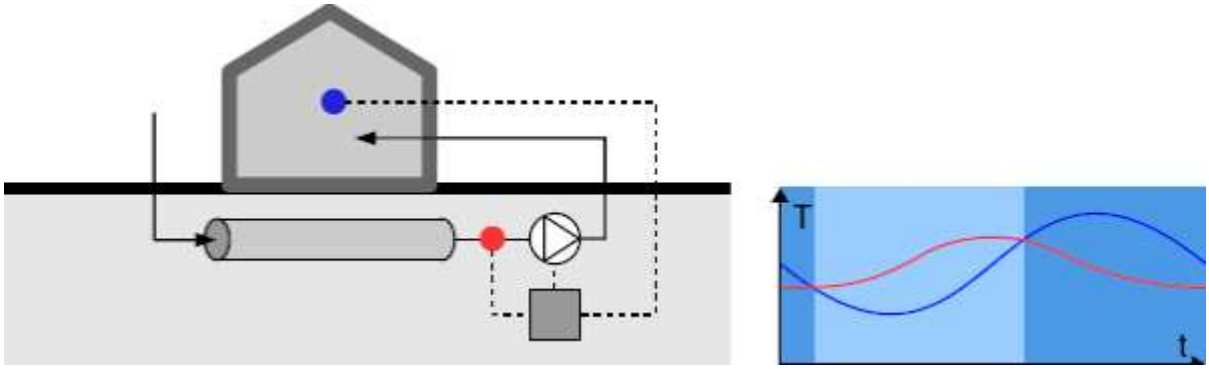


Figure 7 : régulation sonde T sortie tubes – T int.

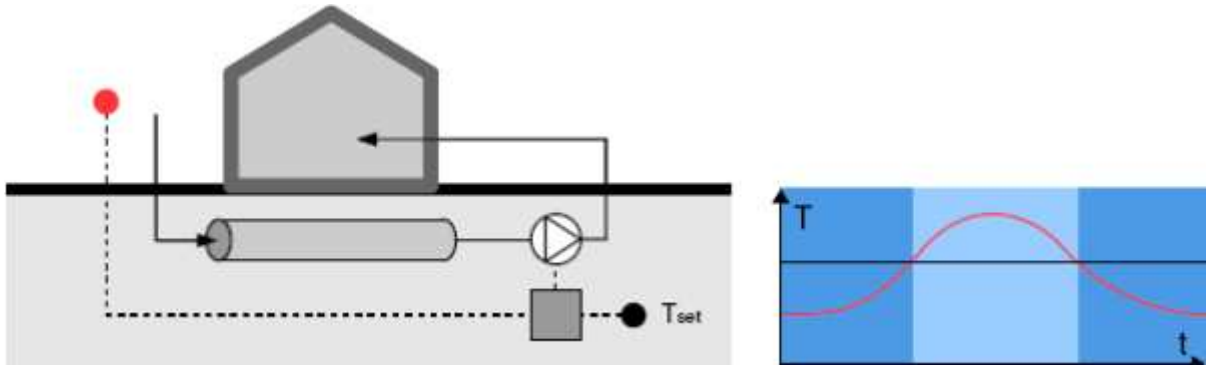


Figure 8 : régulation sonde T ext. – T setpoint

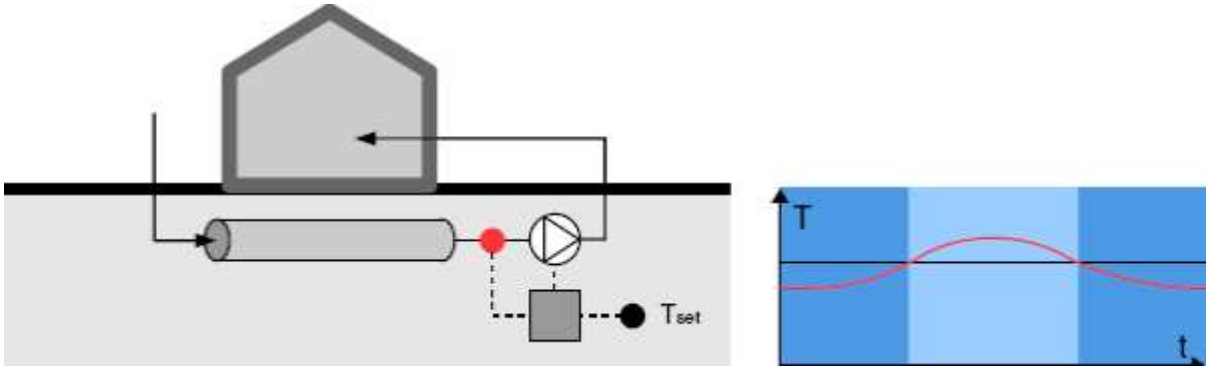


Figure 9 : régulation sonde T sortie tubes – T setpoint

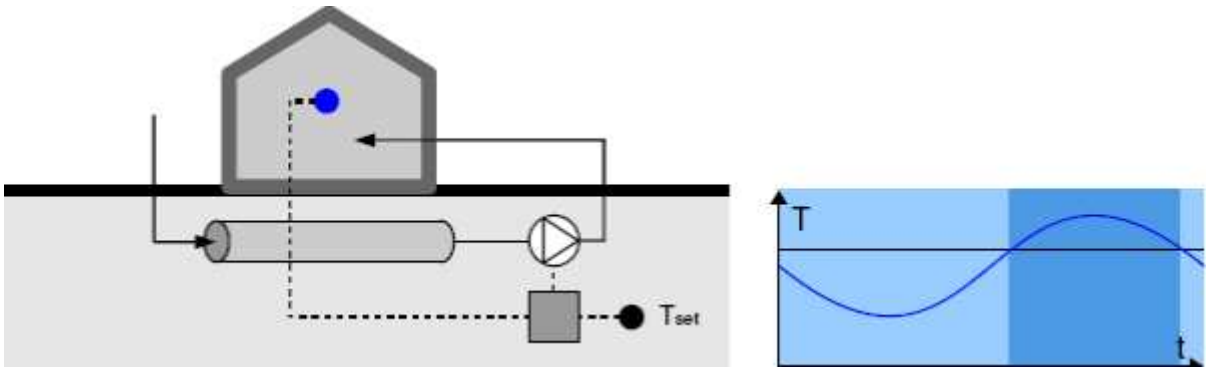


Figure 10 : régulation T int. – T setpoint

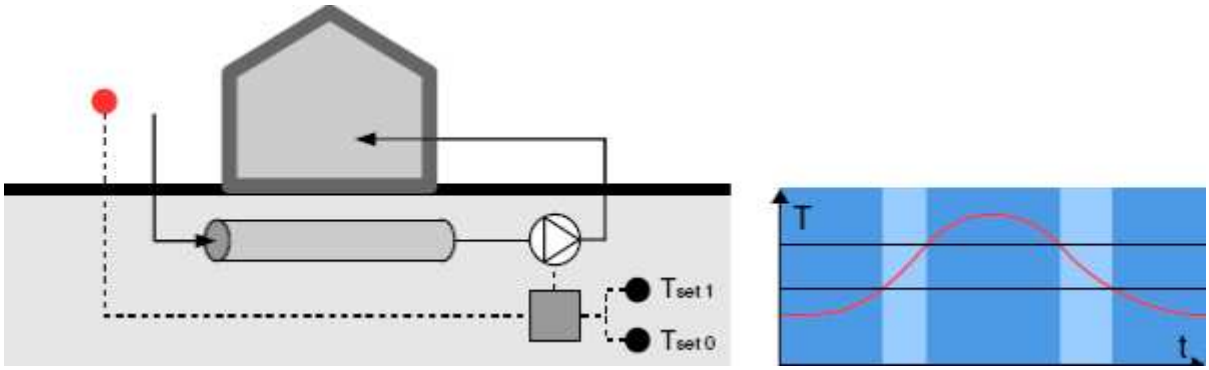


Figure 11 : régulation T ext. - T setpoint inf. – T setpoint sup.

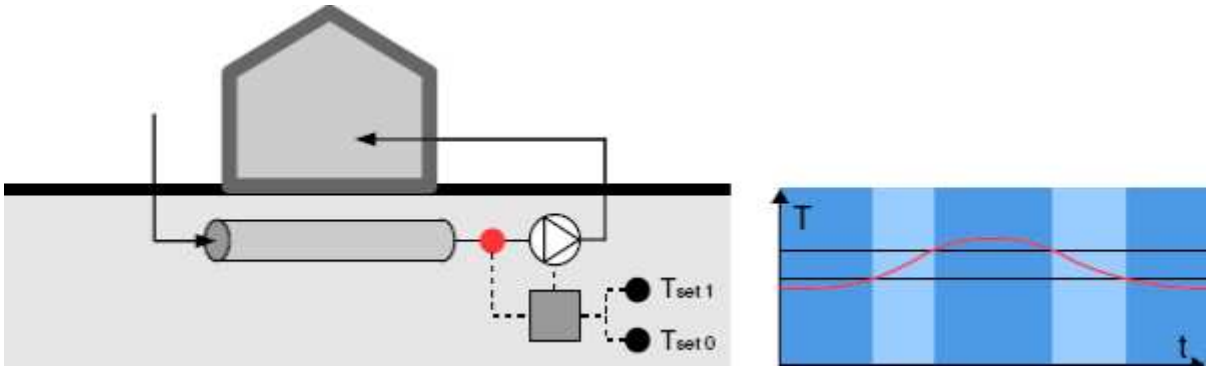


Figure 12 : régulation T sortie tubes – T setpoint inf – T setpoint sup.

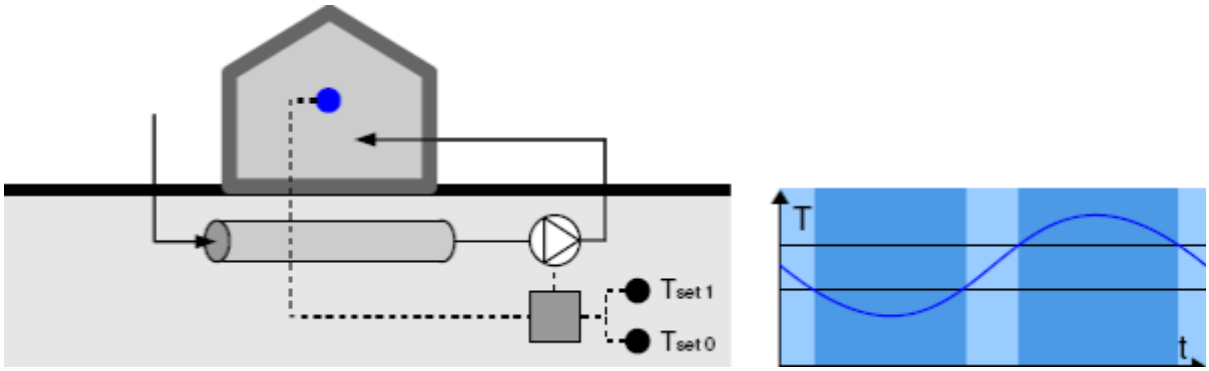


Figure 13 : régulation T int.  – T setpoint inf – T setpoint sup.

## Onglet ‘Simulation’

Cet onglet définit les conditions cadre de la simulation :

– Section typique ou

– Modèle complet,

… et permet de démarrer la simulation.

Selon la complexité du modèle et les ressources matérielles de l’ordinateur, le temps de calcul peut être d’une minute à une heure, environ. L’avancement de la simulation est matérialisé par une barre de progression.

## Fenêtre ‘Console’

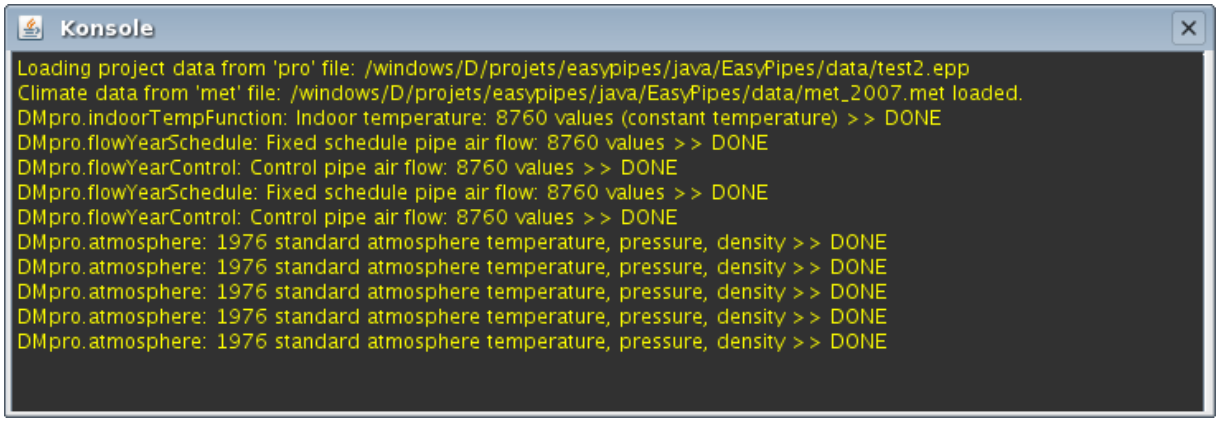


Figure 14 : capture fenêtre console

La console donne un feed-back sur les différentes opérations et fonctions utilisées par le programme. Elles permettent à l’utilisateur de vérifier si les donnes saisies sont pertinentes et correctement pris en compte par le programme.

## Fenêtre ‘Results’ (output)

Les résultats de la simulation sont enregistrés dans un fichier horaire pouvant être analysé à la demande par l'utilisateur. L’interface graphique propose une analyse rapide qui donne un feed-back de simulation immédiat:

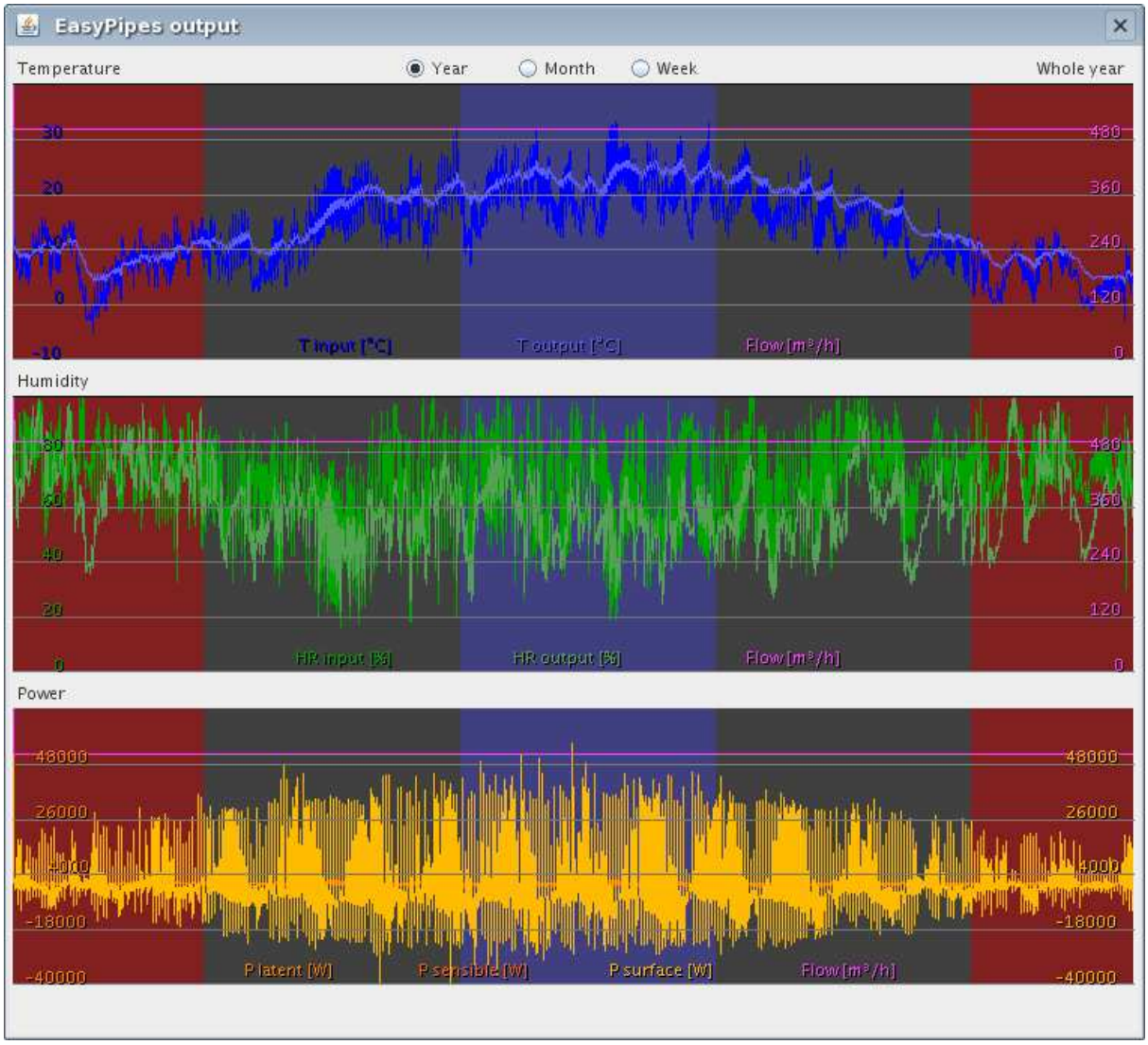


Figure 15 : capture fenêtre résultats

La fenêtre de résultats est disponible dès que la simulation est terminée. Elle permet une première analyse des résultats en affichant par intervalle d’une année, mois, semaine, les valeurs horaires suivantes :

- Température et humidité relative de l’air à l’entrée de tubes (℃ ; %)
- Température et humidité relative de l’air à la sortie des tubes (℃ ; %)
- Flux de chaleur (puissances ; W) :
    - Latente,
    - Sensible
    - Par la surface
- Débit d’air (m³/h)

Pour une analyse approfondie selon des critères individuels, l’utilisateur peut se référer à un fichier de résultats au format ‘CSV’.

## Connexion avec le ‘Type 460’

Quand l’utilisateur demande l’exécution de la simulation, l’interface génère et met à jour l’ensemble des données requis pour la simulation, écrit deux fichiers d’échange de données et exécute la DLL TRNSYS de simulation précompilée.

Les paramètres qui décrivent le modèle sont enregistrés dans un fichier *.dck. C’est le fichier requis par la routine précompilée de TRNSYS pour paramétrer la simulation. Les données horaires requis pour la simulation sont stockées dans un fichier data au format ‘CSV’.

L’exécutable de simulation va générer un fichier de résultats horaire au format ‘CSV’. Le contenu de ce fichier est affiché dans la fenêtre de résultats, directement à partir de l’interface graphique.

Comme TRNSYS, les routines sont dépendantes du système d’exploitation MS Windows (XP ou Vista).

## Prérequis informatiques

- Plateforme MS Windows XP ou Vista
- Machine virtuelle JAVA

# Conclusions

Débuté fin 2007, ce projet a été mené conjointement par le CUEPE – Institut des Sciences de l'Environnement de l'Université de Genève, et par l'Université de Lisbonne, du fait du transfert de Pierre Hollmuller, responsable du module TRNSYS, dans cette Université.

Cette collaboration fructueuse a permis de mener à bien ce développement et de finaliser une version β de EasyPipes.

Dans un premier temps la version β sera distribuée à un groupe d'utilisateurs confidentiels pour tester son fonctionnement et son ergonomie.

Après un « nettoyage » et quelques améliorations suggérées par ces essais, EasyPipes sera distribué, suivi et adapté de manière continue pour répondre aux besoins des milieux académiques et professionnels.

# Annexe

## Documentation Routine Trnsys

core

# Class DMpro

```
java.lang.Object
  └core.DMpro
```

```
public class DMpro
extends java.lang.Object
```

This class contains all project data and methods to calculate h/h schedules, get and set methods

## Constructor Summary

| | |
|---|---|
| `DMpro()` | Default constructor |

## Method Summary

| | |
|---|---|
| `double[]` | `atmosphere(double Alt)` This method computes air -temperature, -pressure and -desity for given altitude according to U.S. 1976 Standard Atmosphere model |
| `double[]` | `boundary()` This method returns h/h equivalent boundary temperature Formula: Tb[hh] = T ext[hh] + alpha * R * Gh[hh] |
| `java.lang.String` | `composeFileString(java.lang.String txt, java.lang.String cTag)` This method composes and appends project data to the string required to save user data to file. |
| `double` | `flowController(int controlFlag, double tSet0, double tSet1, double Flow0, double Flow1, double Hyst, int hh)` This is the controller for air flow through pipes: returns air flow depending on controller input for selected hour of year |
| `double[]` | `flowYearControl()` This method evaluates air flow through pipes with controller regulation for whole year |
| `double[]` | `flowYearSchedule()` This method evaluates air flow through pipes with fixed schedule for whole year |
| `double[][]` | `getControlFlow()` This function returns h/h controlFlow |

| | |
|---|---|
| double[] | **getControlHysteresis**()<br>This function returns h/h controlHysteresis |
| double[][] | **getControlSetpoint**()<br>This fuction returns h/h control setpoints |
| int | **getCool**(int be)<br>Returns the hour of the year of the biginning or the ending of the cooling season. |
| double[] | **getCoolMFFlowByArray**()<br>Returns the list of the hours of the cooling season, monday to friday. |
| int[] | **getCoolMFTimeByArray**()<br>Returns the list of the hours of the cooling season, monday to friday. |
| double[] | **getCoolSSFlowByArray**()<br>Returns the list of the hours of the cooling season, saturday and sunday. |
| int[] | **getCoolSSTimeByArray**()<br>Returns the list of the hours of the cooling season, saturday and sunday. |
| double[][] | **getData**()<br>Returns data content. |
| double[][] | **getData**(int[][] col)<br>Returns data content. |
| java.lang.String[] | **getdFormat**()<br>Return dFormat content. |
| double | **getEstimatedAirFlowThroughPipes**()<br>Returns the estimated air flow through the pipes |
| int | **getHeat**(int be)<br>Returns the hour of the year of the biginning or the ending of the heating season. |
| double[] | **getHeatMFFlowByArray**()<br>Returns the list of the hours of the heating season, monday to friday. |
| int[] | **getHeatMFTimeByArray**()<br>Returns the list of the hours of the heating season, monday to friday. |
| double[] | **getHeatSSFlowByArray**()<br>Returns the list of the hours of the heating season, saturday and sunday. |
| int[] | **getHeatSSTimeByArray**()<br>Returns the list of the hours of the heating season, saturday and sunday. |
| int | **getHourOfYear**(java.lang.Object object)<br>Returns the hour of the year of "date". |

| | |
|---|---|
| double[] | **getIndoorTemp**()<br>This method returns h/h building indoor temperature profile |
| double[] | **getMidsMFFlowByArray**()<br>Returns the list of the hours of the mid-season, monday to friday. |
| int[] | **getMidsMFTimeByArray**()<br>Returns the list of the hours of the mid-season, monday to friday. |
| double[] | **getMidsSSFlowByArray**()<br>Returns the list of the hours of the mid-season, saturday and sunday. |
| int[] | **getMidsSSTimeByArray**()<br>Returns the list of the hours of the mid-season, saturday and sunday. |
| int | **getNumberOfPipes**()<br>Returns the number of the pipes. |
| double[] | **getPipeFlow**()<br>This method returns h/h flow through buried pipes for whole year |
| int[] | **getSeason**()<br>This function returns h/h season flag |
| int[] | **getSeasons**() |
| double | **getTotalLengthOfPipes**()<br>Returns the total length of the pipes. |
| double | **getTotalSurfaceOfPipes**()<br>Returns the total boundary of the pipes. |
| double | **getVar**(java.lang.String varName, double value)<br>This method gets the double value 'varName' |
| int | **getVar**(java.lang.String varName, int value)<br>This method gets the int value 'varName' |
| java.lang.String | **getVar**(java.lang.String varName, java.lang.String value)<br>This method gets the String 'varName' |
| double[] | **listToArray**(java.lang.String list)<br>This method converts a comma separated list into double[] |
| void | **loadProjectData**(java.util.Properties properties)<br>This method reads user project file and loads data into variables |
| double[][] | **pipeIntake**()<br>This method returns h/h Temp[°C] and RH[%] for pipe air intake from climate data or file depending on pipeFlag. |
| int | **refreshData**()<br>This method updates all project and simulation data and loads hourly data it into the data[][] variable |

| | |
|---|---|
| void | **setCool**(int be, int hourOfDay)<br>Sets the hour of the year of the biginning or the ending of the cooling season. |
| void | **setCool**(int be, java.lang.Object date)<br>Sets the hour of the year of the biginning or the ending of the cooling season. |
| void | **setData**(double[] dat, int column)<br>Load h/h data in dat into data array<br><br>Columns:<br>[0] : time [hour of year: 1...8760]<br>[1] : season [0 = heat; 1 = mids; 2 = cool] [2] : T air ext [°C]<br>[3] : HR air ext [%]<br>[4] : P atmospheric pressure [Pa]<br>[5] : Gh [W/m²]<br>[6] : T inside building [°C]<br>[7] : T boundary [°C]<br>[8] : T input pipe [°C]<br>[9] : HR input pipe [%]<br>[10] : airflow pipe [m³/h]<br>[11] : airflow controlled high [m³/h]<br>[12] : airflow controlled low [m³/h]<br>[13] : control setpoint up [°C]<br>[14] : control setpoint down [°C]<br>[15] : control hysteresis [K] |
| void | **setHeat**(int be, int hourOfDay)<br>Sets the hour of the year of the biginning or the ending of the heating season. |
| void | **setHeat**(int be, java.lang.Object date)<br>Sets the hour of the year of the biginning or the ending of the heating season. |
| void | **setVar**(java.lang.String varName, double value)<br>This method sets the variable 'varName' to the double 'value' |
| void | **setVar**(java.lang.String varName, float value)<br>This method sets the variable 'varName' to the float 'value' |
| void | **setVar**(java.lang.String varName, int value)<br>This method sets the variable 'varName' to the integer 'value' |
| void | **setVar**(java.lang.String varName, long value)<br>This method sets the variable 'varName' to the long 'value' |
| void | **setVar**(java.lang.String varName, java.lang.Object value)<br>This method sets the variable 'varName' to the Object 'value' |
| void | **setVar**(java.lang.String varName, java.lang.String value)<br>This method sets the variable 'varName' to the String 'value' |

| **Methods inherited from class java.lang.Object** |
|---|
| clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

# Constructor Detail

## DMpro

```
public DMpro()
```

Default constructor

# Method Detail

## refreshData

```
public int refreshData()
```

This method updates all project and simulation data and loads hourly data it into the data[][] variable

**Returns:**

int result: -1 if error

---

## setData

```
public void setData(double[] dat,
                    int column)
```

Load h/h data in dat into data array

Columns:
[0] : time [hour of year: 1...8760]
[1] : season [0 = heat; 1 = mids; 2 = cool] [2] : T air ext [°C]
[3] : HR air ext [%]
[4] : P atmospheric pressure [Pa]
[5] : Gh [W/m²]
[6] : T inside building [°C]
[7] : T boundary [°C]
[8] : T input pipe [°C]
[9] : HR input pipe [%]
[10] : airflow pipe [m³/h]
[11] : airflow controlled high [m³/h]
[12] : airflow controlled low [m³/h]
[13] : control setpoint up [°C]
[14] : control setpoint down [°C]
[15] : control hysteresis [K]

**Parameters:**

`dat` - double[]: hourly data
`column` - int: datta array target column

---

## getVar

```
public double getVar(java.lang.String varName,
                     double value)
```

This method gets the double value 'varName'

**Parameters:**

`varName` - String: variable name
`value` - double: a random double value to identify type

---

## getVar

```
public int getVar(java.lang.String varName,
                  int value)
```

This method gets the int value 'varName'

**Parameters:**

`varName` - String: variable name
`value` - int: a random int value to identify type

---

## getVar

```
public java.lang.String getVar(java.lang.String varName,
                               java.lang.String value)
```

This method gets the String 'varName'

**Parameters:**

`varName` - String: variable name
`value` - String: a random string to identify type

---

## setVar

```
public void setVar(java.lang.String varName,
                   java.lang.String value)
```

This method sets the variable 'varName' to the String 'value'

**Parameters:**

`varName` - String: variable name
`value` - String: String

---

## setVar

```
public void setVar(java.lang.String varName,
                   double value)
```

This method sets the variable 'varName' to the double 'value'

**Parameters:**

`varName` - String: variable name
`value` - double: double value

---

## setVar

```
public void setVar(java.lang.String varName,
                   float value)
```

This method sets the variable 'varName' to the float 'value'

**Parameters:**

`varName` - String: variable name
`value` - float: float value

---

## setVar

```
public void setVar(java.lang.String varName,
                   int value)
```

This method sets the variable 'varName' to the integer 'value'

**Parameters:**

`varName` - String: variable name
`value` - int: integer value

---

## setVar

```
public void setVar(java.lang.String varName,
                   long value)
```

This method sets the variable 'varName' to the long 'value'

**Parameters:**

`varName` - String: variable name
`value` - long: double value

---

## setVar

```
public void setVar(java.lang.String varName,
                   java.lang.Object value)
```

This method sets the variable 'varName' to the Object 'value'

**Parameters:**

`varName` - String: variable name
`value` - Object: object containing a String, Double or Integer

---

## boundary

```
public double[] boundary()
```

This method returns h/h equivalent boundary temperature
Formula: Tb[hh] = T ext[hh] + alpha * R * Gh[hh]

**Returns:**
Tb double[]: equiv Temp [°C]

---

## pipeIntake

```
public double[][] pipeIntake()
```

This method returns h/h Temp[°C] and RH[%] for pipe air intake from climate data or file depending on pipeFlag.

**Returns:**
result double[][]: result >> [0] = temperature [°C], [1] = HR [%]

---

## flowYearSchedule

```
public double[] flowYearSchedule()
```

This method evaluates air flow through pipes with fixed schedule for whole year

**Returns:**
double[] flowYear: array containing 8760 hourly flow values [m³/h]

---

## listToArray

```
public double[] listToArray(java.lang.String list)
```

This method converts a comma separated list into double[]

**Parameters:**
`list` - String: e.g. "0,6,12,14,18"
**Returns:**
result double[]: array containing converted double values

---

## getPipeFlow

```
public double[] getPipeFlow()
```

This method returns h/h flow through buried pipes for whole year

**Returns:**
pipeFlow double[]: flow [m³/h]

---

## flowYearControl

```
public double[] flowYearControl()
```

This method evaluates air flow through pipes with controller regulation for whole year

**Returns:**

double[] flowYear: array containing 8760 hourly flow values [m³/h]

---

## flowController

```
public double flowController(int controlFlag,
                             double tSet0,
                             double tSet1,
                             double Flow0,
                             double Flow1,
                             double Hyst,
                             int hh)
```

This is the controller for air flow through pipes: returns air flow depending on controller input for selected hour of year

**Parameters:**

`controlFlag` - int: control type [0...7]
`tSet0` - double: setpoint temperature 1 [°C]
`tSet1` - double: setpoint temperature 2 [°C]
`Flow0` - double: flow if condition = true [m³/h]
`Flow1` - double: flow if condition = false [m³/h]
`Hyst` - double: hysteresis [°C]
`hh` - int: hour of year -1

**Returns:**

flow double: airflow [m³/h] (HIGH ou LOW)

---

## getControlFlow

```
public double[][] getControlFlow()
```

This function returns h/h controlFlow

**Returns:**

double[][]: controlFlow (hh)(0/1) [m³/h]

---

## getControlSetpoint

```
public double[][] getControlSetpoint()
```

This fuction returns h/h control setpoints

**Returns:**

double[][]: controlsetpoint (hh)(0/1) [°C]

---

## getControlHysteresis

```
public double[] getControlHysteresis()
```

This function returns h/h controlHysteresis

**Returns:**
double[]: hysteresis (hh) [°C]

---

## getSeason

```
public int[] getSeason()
```

This function returns h/h season flag

**Returns:**
double[]: season [0 = heat; 1 = mids; 2 = cool]

---

## getIndoorTemp

```
public double[] getIndoorTemp()
```

This method returns h/h building indoor temperature profile

**Returns:**
double[]: temperature (hh) [°C]

---

## atmosphere

```
public double[] atmosphere(double Alt)
```

This method computes air -temperature, -pressure and -desity for given altitude according to U.S. 1976 Standard Atmosphere model

**Parameters:**
`Alt` - double: altitude [m] above sea level (range from 0 to 84000m)
**Returns:**
double[]: result >> [0] = temperature [°C], [1] = pressure [Pa], [2] = density [kg/m³]

---

## getCool

```
public int getCool(int be)
```

Returns the hour of the year of the biginning or the ending of the cooling season.

**Parameters:**
`be` - int:

- 0: Beginning of the cooling season
- 1: Ending of the cooling season

**Returns:**

int: Hour of the year

---

## setCool

```
public void setCool(int be,
                    int hourOfDay)
```

Sets the hour of the year of the biginning or the ending of the cooling season.

**Parameters:**

`be` - int:
- 0: Beginning of the cooling season
- 1: Ending of the cooling season

`hourOfDay` - int: Hour of the year

---

## setCool

```
public void setCool(int be,
                    java.lang.Object date)
```

Sets the hour of the year of the biginning or the ending of the cooling season.

**Parameters:**

`be` - int:
- 0: Beginning of the cooling season
- 1: Ending of the cooling season

`date` - Object: Hour of the year

---

## getHeat

```
public int getHeat(int be)
```

Returns the hour of the year of the biginning or the ending of the heating season.

**Parameters:**

`be` - int:
- 0: Beginning of the heating season
- 1: Ending of the heating season

**Returns:**

int: Hour of the year

---

## setHeat

```
public void setHeat(int be,
                    int hourOfDay)
```

Sets the hour of the year of the biginning or the ending of the heating season.

**Parameters:**

`be` - int:

- 0: Beginning of the heating season
- 1: Ending of the heating season

`hourOfDay` - int: Hour of the year

---

## setHeat

```
public void setHeat(int be,
                    java.lang.Object date)
```

Sets the hour of the year of the biginning or the ending of the heating season.

**Parameters:**

`be` - int:

- 0: Beginning of the heating season
- 1: Ending of the heating season

`date` - Object: Hour of the year

---

## getNumberOfPipes

```
public int getNumberOfPipes()
```

Returns the number of the pipes.

**Returns:**

int: Number of the pipes [n]

---

## getTotalLengthOfPipes

```
public double getTotalLengthOfPipes()
```

Returns the total length of the pipes.

**Returns:**

double: Total length of the pipes [m]

---

## getTotalSurfaceOfPipes

```
public double getTotalSurfaceOfPipes()
```

Returns the total boundary of the pipes.

**Returns:**

double: Total boundary of the pipes [m²]

---

## getEstimatedAirFlowThroughPipes

```
public double getEstimatedAirFlowThroughPipes()
```

Returns the estimated air flow through the pipes

**Returns:**
double: Estimated air flow through pipes [m³/h]

---

## getCoolMFTimeByArray

```
public int[] getCoolMFTimeByArray()
```

Returns the list of the hours of the cooling season, monday to friday.

**Returns:**
int[]: Array of the hours of the cooling season.

---

## getCoolMFFlowByArray

```
public double[] getCoolMFFlowByArray()
```

Returns the list of the hours of the cooling season, monday to friday.

**Returns:**
int[]: Array of the hours of the cooling season.

---

## getCoolSSTimeByArray

```
public int[] getCoolSSTimeByArray()
```

Returns the list of the hours of the cooling season, saturday and sunday.

**Returns:**
int[]: Array of the hours of the cooling season.

---

## getCoolSSFlowByArray

```
public double[] getCoolSSFlowByArray()
```

Returns the list of the hours of the cooling season, saturday and sunday.

**Returns:**
int[]: Array of the hours of the cooling season.

---

## getMidsMFTimeByArray

```
public int[] getMidsMFTimeByArray()
```

Returns the list of the hours of the mid-season, monday to friday.

**Returns:**
int[]: Array of the hours of the mid-season.

---

## getMidsMFFlowByArray

```
public double[] getMidsMFFlowByArray()
```

Returns the list of the hours of the mid-season, monday to friday.

**Returns:**
int[]: Array of the hours of the mid-season.

---

## getMidsSSTimeByArray

```
public int[] getMidsSSTimeByArray()
```

Returns the list of the hours of the mid-season, saturday and sunday.

**Returns:**
int[]: Array of the hours of the mid-season.

---

## getMidsSSFlowByArray

```
public double[] getMidsSSFlowByArray()
```

Returns the list of the hours of the mid-season, saturday and sunday.

**Returns:**
int[]: Array of the hours of the mid-season.

---

## getHeatMFTimeByArray

```
public int[] getHeatMFTimeByArray()
```

Returns the list of the hours of the heating season, monday to friday.

**Returns:**
int[]: Array of the hours of the heating season.

---

## getHeatMFFlowByArray

```
public double[] getHeatMFFlowByArray()
```

Returns the list of the hours of the heating season, monday to friday.

**Returns:**
int[]: Array of the hours of the heating season.

---

## getHeatSSTimeByArray

```
public int[] getHeatSSTimeByArray()
```

Returns the list of the hours of the heating season, saturday and sunday.

**Returns:**
int[]: Array of the hours of the heating season.

---

## getHeatSSFlowByArray

```
public double[] getHeatSSFlowByArray()
```

Returns the list of the hours of the heating season, saturday and sunday.

**Returns:**
int[]: Array of the hours of the heating season.

---

## getHourOfYear

```
public int getHourOfYear(java.lang.Object object)
```

Returns the hour of the year of "date".

**Parameters:**
`date` - Date: Date to be converted to hour of year.
**Returns:**
int: Hour of the year of the "date".

---

## getSeasons

```
public int[] getSeasons()
```

---

## getData

```
public double[][] getData()
```

Returns data content.

**Returns:**
double[][]: Data content.

---

## getdFormat

```
public java.lang.String[] getdFormat()
```

Return dFormat content.

**Returns:**
String[]: dFormat content.

---

## getData

```
public double[][] getData(int[][] col)
```

Returns data content.

**Parameters:**
`col` - int[][]: Source and target column number(s) to read (maximum 10):

| | |
|---|---|
| Nr. of source column | Nr. of target column |
| Nr. of source column | Nr. of target column |
| ... | ... |

**Returns:**
double[][]: Data content.

---

## loadProjectData

```
public void loadProjectData(java.util.Properties properties)
```

This method reads user project file and loads data into variables

**Parameters:**
`properties` -

---

## composeFileString

```
public java.lang.String composeFileString(java.lang.String txt,
                                          java.lang.String cTag)
```

This method composes and appends project data to the string required to save user data to file.

**Parameters:**
`txt` - String: String with text to append to
`cTag` - String: tag used for comments ('#' recommended)
**Returns:**
txt String

---

# Annexe

## Documentation EasyPipes (Java)

# TYPE 460: AIR-SOIL HEAT EXCHANGER

## General Description

This component models an air--soil heat exchanger. It accounts for sensible as well as latent exchanges between airflow and pipes, diffusion into surrounding soil, frictional losses and possible water infiltration into the pipes. Direction of airflow can be reversed (stratification in case of heat storage) and flexible geometry allows for inhomogenous soils as well as diverse border conditions.

This type can be used in standard or expert mode.

In standard mode, which covers most of the situations (array of circular pipes within a homogeneous soil, adiabatic lateral conditions), the limited number of parameters are defined as routine as arguments. In expert mode, which allows handling of complex situations (non homogeneous soils) as well as for definition of optional outputs, the parameters are read from a parameter definition file, which has to be edited manually.

## Mathematical Description

Hypothesis

Following hypothesis have been adopted:

- So as to be flexible, the orthogonal meshing allows for variable node widths in all three dimensions. Circular tubes are represented by way of equivalent square sections, lateral exchange surface being computed by way of an adequate corrective factor.
- Thermal heat diffusion is fully three dimensional. Soil characteristics may be inhomogeneous but are constant in time.
- Border conditions, which may be various on the same face, are either adiabatic or driven by a transient input. Latter can be defined in terms of temperature or heat load, with possibility to include an additional surface resistance.
- As for most other model and coherent with analytical approach developed in parallel (Hollmuller 2003), air temperature and velocity are considered to be uniform within a pipe section. Heat exchange with pipe is treated by means of an overall convective which depends on velocity, but not on temperature.
- The thermal effect of the charge losses, computed in function of a friction factor, the tube surface and the air velocity, is evenly distributed along the tubes. Eventual singular charge losses have to be treated apart.
- Transient water infiltration, if any, occurs on a predefined part of the tubes, where it adds to possible condensed water.

Algorithm

The model’s kernel bases on the energy and mass exchanges between the airflow and the pipe. They are computed iteratively for each pipe node, from air inlet to outlet, and comprise following iterms.

The sensible heat lost by the airflow:

$$P_{sbl} = S_{tub} \cdot h \cdot \left(T_{air} - T_{tub}\right)$$

The latent heat, determined by the Lewis analogy, which actually considers former sensible heat to result from a convective air exchange between the flow and a superficial layer at pipe's temperature, the analogy implying following convective air exchange rate:

$$\dot{m}_{conv} = \frac{P_{sbl}}{c_{air} \cdot \left(T_{air} - T_{tub}\right)}$$

Considering the air layer to be saturated in humidity, this air exchange also induces a water vapor exchange, which is determined by the difference in humidity ratios of main flow and superficial layer:

$$\dot{m}_{lat} = \left(W_{air} - W_{tub}\right) \cdot \dot{m}_{conv}$$

where, according to perfect gases:

$$W_{air} = \frac{H \cdot \mathrm{Pr}_{sat}\left(T_{air}\right) \cdot M_{wat}}{\mathrm{Pr}_{air} \cdot M_{air}}$$

$$W_{tub} = \frac{100\% \cdot \mathrm{Pr}_{sat}\left(T_{tub}\right) \cdot M_{wat}}{\mathrm{Pr}_{air} \cdot M_{air}}$$

When positive, this vapor transfer corresponds to condensation, when negative to evaporation. In latter case it is further limited by the free water content in the considered node, as well as by the maximum humidity (saturation pressure) which can be absorbed by the airflow. With these definitions, the associated latent heat finally writes as:

$$P_{lat} = c_{lat} \cdot \dot{m}_{lat}$$

The heat diffusion from the 4 lateral soil nodes and the 2 preceding and following pipe nodes:

$$P_{diff} = \sum_{soil} S_i k_i \left(T_{soil,i,t-1} - T_{tub}\right) + \sum_{tube} S_i k_i \left(T_{tub,i,t-1} - T_{tub}\right)$$

The saturation pressure being non-linear in terms of temperature, the value of Ttub as well as preceding heat rates are being determined by iterative resolution of the energy balance:

$$P_{int} - \left(P_{sbl} + P_{lat} + P_{diff}\right) = 0$$

where the capacitive gains of the pipe and the free water are given by:

$$P_{int} = \frac{\left(c_{tub} \cdot m_{tub} + c_{wat} \cdot m_{wat,t-1}\right) \cdot \left(T_{tub} - T_{tub,t-1}\right)}{\Delta t}$$

The associated hydric balance on its turn allows to determine the new water content of the node:

$$m_{wat} = m_{wat,t-1} + \left(\dot{m}_{inf} - \dot{m}_{lat}\right) \cdot \Delta t$$

Charge losses are taken in account by way of a friction coefficient f, for which typical values are to be found on a Moody diagram (ASHRAE, Ch.2, 1989):

$$P_{fric} = \dot{m}_{air} \cdot f \cdot \frac{l}{d} \cdot \frac{v_{air}^2}{2}$$

Finally, preceding energy and mass balances yield the air input conditions of the next pipe node:

$$T_{air,i} = T_{air} + \frac{P_{fric} - P_{sbl}}{\left(c_{air} + c_{vap} \cdot W_{air}\right) \cdot \dot{m}_{air}}$$

where computation repeats in the same manner.

After completing this calculation for all tube nodes, computation treats diffusion of heat into soil nodes, taking into account user-specified border conditions.

---

## Component Configuration

### *Standard Mode*

Parameters

| Number | Symbol | Definition and unit | |
|---|---|---|---|
| 1 | *Igeo* | geometry type [-] | 1) |
| 2 | *IfileLog* | parameter log file [-] | |
| 3 | *TypAir* | airflow type [-] | 2) |
| 4 | *NtubY* | number of pipes per layer [y axis] [-] | |
| 5 | *NtubZ* | number of superposed pipe layers [z axis] [-] | |
| 6 | *LsoilY* | pipe - pipe inter axial distance, y axis [m] | |
| 7 | *LsoilZ* | pipe - pipe inter axial distance, z axis [m] | |
| 8 | *LsoilTop* | top surface - first pipe layer [m] | |
| 9 | *LsoilBot* | last pipe layer - bottom surface [m] | |
| 10 | *LsoilSide* | lateral surface - first pipe [m] | |
| 11 | *Ltub* | pipe length [m] | |
| 12 | *Dtub* | pipe diameter [m] | |
| 13 | *ThTub* | pipe thickness [m] | |
| 14 | *LamTub* | pipe conductivity [W/K.m] | |
| 15 | *CvTub* | pipe capacity [kJ/K.m3] | |
| 16 | *CtubFric* | pipe friction coefficient [-] | 3) |
| 17 | *LamSoil* | soil conductivity [W/K.m] | |
| 18 | *CvSoil* | soil capacity [kJ/K.m3] | |
| 19 | *TiniSoil* | soil initial temperature [C] | |
| 20 | *TypSurf(1)* | surface type, top [-] | 4) |
| 21 | *Rsurf(1)* | surface resistance, top [K.m2/W] | 5) |
| 22 | *TypSurf(2)* | surface type, bottom [-] | 4) |
| 23 | *Rsurf(2)* | surface resistance, bottom [K.m2/W] | 5) |

Inputs

| Number | Symbol | Definition and unit | |
|---|---|---|---|
| 1 | *XairTot* | Airflow, total over all pipes [m3/hr or kg/h] | 2) 6) |
| 2 | *TairIn* | inlet temperature [C] | |

| | | | |
|---|---|---|---|
| 3 | *HrelIn* | inlet relative humidity [pcent] | |
| 4 | *PrAir* | inlet air pressure [Pa] | |
| 5 | *MwatInfTot* | water infiltration, total over all pipes [kg/h] | |
| 6 | *Xsurf(1)* | surface input, top [C or kJ/hr] | 4) |
| 7 | *Xsurf(2)* | surface input, bottom [C or kJ/hr] | 4) |

Outputs

| **Number** | **Symbol** | **Definition and unit** | |
|---|---|---|---|
| 1 | *TairOut* | outlet temperature [C] | |
| 2 | *HrelOut* | outlet relative humidity [pcent] | |
| 3 | *PsblTot* | sensible energy rate lost by airflow, total over all pipes [W] | |
| 4 | *PlatTot* | latent energy rate lost by airflow, total over all pipes [W] | |
| 5 | *Xbord(1)* | surface output, top [C or W] | 4) |
| 6 | *Xbord(2)* | surface output, bottom [C or W] | 4) |

Explanatory notes for preceeding tables

1) *Igeo* specifies the type of geometry to be simulated:
   0: no pipes (only heat diffusion in relation with top/bottom surface input)
   1: typical pipe module (one pipe per layer, with adiabatic conditions at inter-axial distance), disregarding lateral border effects.
   2: entire pipe array, including lateral border effects. Beware that this setting can be very more time consuming.
2) Parameter *TypeAir* specifies the type of airflow:
   0: *XairTot* is a volumetric flow [m3/h]
   1: *XairTot* is a mass flow [kg/h]
3) Typical value of friction coefficient is 0.01 - 0.02 [-].
4) For each surface, surface type is one of the following :
   0 : input *Xsurf* is surface temperature, output *Xbord* is corresponding inflowing energy rate.
   1 : input *Xsurf* is inflowing energy rate, output *Xbord* is corresponding equivalent border temperature (first soil layer)
5) A negative value of *Rsurf* indicates an adiabatic surface. In this case corresponding *TypSurf* should be set to 0.
6) So as to avoid oscillations of air temperature along the tube, airflow should not exceed a minimum value X*airMin*, which is written to the parameter log file. An airflow smaller than this minmum value will be set to zero.

***Expert Mode***

Parameters

| **Number** | **Symbol** | **Definition and unit** |
|---|---|---|
| 1 | *IfileDef* | parameter definition file [-] |
| 2 | *IfileLog* | parameter log file [-] |

Inputs

| Number | Symbol | Definition and unit |
|---|---|---|
| 1 | *XairTot* | airflow, total over all pipes [m3/hr or kg/h] |
| 2 | *TairIn* | inlet temperature [C] |
| 3 | *HrelIn* | inlet relative humidity [pcent] |
| 4 | *PrAir* | inlet air pressure [Pa] |
| 5 | *MwatInfTot* | water infiltration, total over all pipes [kg/h] |
| 6 | *Xsurf(1)* | surface input, first [C or W] |
| … | … | … |
| 5+Nsurf | *Xsurf(Nsurf)* | surface input, last [C or W] |

Outputs

| Number | Symbol | Definition and unit |
|---|---|---|
| 1 | *TairOut* | outlet temperature [C] |
| 2 | *HrelOut* | outlet relative humidity [pcent] |
| 3 | *PsblTot* | sensible energy rate lost by airflow, total over all pipes [W] |
| 4 | *PlatTot* | latent energy rate lost by airflow, total over all pipes [W] |
| 5 | *Xbord(1)* | surface output, first [C or W] |
| … | … | … |
| 4+Nsurf | *Xbord(Nsurf)* | surface output, last [C or W] |
| 4+Nsurf+1 | *Xopt(1)* | optional output, first |
| … | … | … |
| 4+Nsurf +Nopt | *Xopt(Nopt)* | optional output, last |

Parameter definition file

Each data set hereafter is written on one line (exception for *TypSoil* arrays, which take *NK* or *NK*+2 lines). Data within one dataset is separated by commas or blanks. Comments can be entered by using an asterix (*) in first column.

| Symbol | Definition and unit |
|---|---|
| *TypAir* | airflow type [-] |
| *Dt, XairTotMin, TtubErr* | internal timestep [h], minimum airflow [m3/h or kg/h], tolerance on tube temperature [C] |
| *Nmod,Nsec,Nsoil,Nsurf,NI,NJ, NK* | number of : modules, cross-sections, surfaces, nodes along x-axis, nodes along y-axis, nodes along z-axis [-] |
| *DX (1:NI)* | node width along x-axis [m] |
| *DY (1:NJ)* | node width along y-axis [m] |
| *DZ (1:NK)* | node width along z-axis [m] |
| *TypSec(1:NI)* | type of used cross-sections along x-axis [-] |
| *TypSoil (1:NJ,1:NK)* | type of surfaces on frontal cross-section [-] |
| *TypSoil (0:NJ+1,0:NK+1)* | type of soils/surfaces for typical cross-section in y-z plane [-] |
| *TypSoil (1:NJ,1:NK)* | type of surfaces on rear cross-section [-] |
| *TypSurf, Rsurf* | surface type [-] and resistance [K.m2/W] |
| *PosInf* | position of water infiltration [-] |

| | |
|---|---|
| *Kair0, Kair1* | air-tube exchange coefficients [W/K.m2] and [W/K.m2 per m/s] |
| *LamSoil, CvSoil* | soil conductivity [W/K.m] and capacity [kJ/K.m3] |
| *LamTub, CvTub* | tube conductivity [W/K.m] and capacity [kJ/K.m3] |
| *ThTub, CtubCor, CtubFric* | tube thickness [m], circumference correction factor [-] and friction coefficient [-] |
| *TypWatFlow (-1:1)* | type of water flow [-] |
| *Vwat (-1:1)* | velocity of water flow [m/h] |
| *NiniSoil, NiniWat* | number of initial conditions (soil temperatures and waterthicknesses) [-] |
| *TiniSoil, PosIniSoil (1:6)* | initial temperature [C] and corresponding node position [-] |
| *ThIniWat, PosIniWat (1:6)* | initial waterthickness [m] and corresponding node position [-] |
| *Nopt* | number of optional outputs |
| *TypOpt, PosOpt (1:6)* | type of optional output [-] and corresponding node position [-] |

Parameter log file

After formatted copy of the parameter definition file, following data is written at the end of the file.

| **Symbol** | **Definition and unit** |
|---|---|
| *Ntub* | number of pipes (per module) [-] |
| *IflowIni* | node index of tube start along x-axis [-] |
| *IflowEnd* | node index of tube end along x-axis [-] |
| *PosTub(1:2)* | node index of tube position along y- and z-axis [-] |
| *LX* | array length, x axis [m] |
| *LY* | array width, y axis (total over modules) [m] |
| *LZ* | depth of hypocaust [m] |
| *Ltub* | length of pipes [m] |
| *SairTot* | pipe cross-section area, total over all pipes and modules [m2] |
| *StubTot* | pube surface, total over all pipes and modules [m2] |
| *DtubTot* | hydraulic diameter of equivalent unique pipe [m] |
| *SinfTot* | water infiltration surface, total over all modules [m2] |
| *Sbord* | border area, total over all modules [m2] |
| *Kbord* | equivalent border conduction coefficient [W/K.m2] |
| *NnodeOpt* | number of nodes concerned by otional output |
| *Dt* | Internal timestep used in simulation [hr] |
| *XairTotMin* | Minimum airflow used in simulation [m3/hr or kg/h] |

Explanatory notes for optional outputs

For each one of them *TypOpt* specifies the type of optional output and takes a value from one of the three following tables. *PosOpt* finally defines the rectangular node cluster for which the optional output is to be considered.

*Optional outputs for tube nodes :*

| **Type** | **Symbol** | **Definition and unit** |
|---|---|---|

| | | | |
|---|---|---|---|
| 1 | *Tair* | Air temperature [C] | * |
| 2 | *Hrel* | Air relative humidity [pcent] | * |
| 3 | *Habs* | Air absolute humidity [Pa] | * |
| 4 | *Hrat* | Air humidity ratio [kg vapor/kg air] | * |
| 5 | *Mwat* | Free water in node [m3] | ** |
| 6 | *MwatLat/Dt* | Water condensing (>0) or evaporating (<0) [kg/h] | ** |
| 7 | *MwatIn/Dt* | Water flowing into node [kg/h] | ** |
| 8 | *MwatInf/Dt* | Water infiltrating into node [kg/h] | ** |
| 9 | *MwatOut/Dt* | Water flowing out of node [kg/h] | ** |
| 10 | *Tsoil* | Tube temperature [C] | ** |
| 11 | *Psbl* | Sensible energy rate from air to tube [W] | ** |
| 12 | *Plat* | Latent energy rate from air to tube [W] | ** |
| 13 | *Pwat* | Energy rate lost by free water [W] | ** |
| 14 | *Pfric* | Energy rate from frictional losses [W] | ** |
| 15 | *Psoil(0)* | Energy rate diffused from all 6 neighbor nodes [W] | ** |
| 16 | *Psoil(1)* | Energy rate diffused from previous neighbor node along x-axis (from surface if border node) [W] | ** |
| 17 | *Psoil(2)* | Energy rate diffused from next neighbor node along x-axis (from surface if border node) [W] | ** |
| 18 | *Psoil(3)* | Energy rate diffused from previous neighbor node along y-axis (from surface if border node) [W] | ** |
| 19 | *Psoil(4)* | Energy rate diffused from next neighbor node along y-axis (from surface if border node) [W] | ** |
| 20 | *Psoil(5)* | Energy rate diffused from previous neighbor node along z-axis (from surface if border node) [W] | ** |
| 21 | *Psoil(6)* | Energy rate diffused from next neighbor node along z-axis (from surface if border node) [W] | ** |
| 22 | *Pint* | Energy rate of internal gains [W] | ** |
| 23 | *Xair* | Air flowrate [m3/h or kg/h] | * |
| 24 | *Vair* | Air velocity [m/s] | * |

* averaged over node cluster

** integrated over node cluster and multiplied by number of modules

*Optional outputs for soil nodes :*

| **Type** | **Symbol** | **Definition and unit** | |
|---|---|---|---|
| 101 | *Tsoil* | Soil temperature [C] | * |
| 102 | *Psoil(0)* | Energy rate diffused from all 6 neighbor nodes [W] | ** |
| 103 | *Psoil(1)* | Energy rate diffused from previous neighbor node along x-axis (from surface if border node) [W] | ** |
| 104 | *Psoil(2)* | Energy rate diffused from next neighbor node along x-axis (from surface if border node) [W] | ** |
| 105 | *Psoil(3)* | Energy rate diffused from previous neighbor node along y-axis (from surface if border node) [W] | ** |
| 106 | *Psoil(4)* | Energy rate diffused from next neighbor node along y-axis (from surface if border node) [W] | ** |
| 107 | *Psoil(5)* | Energy rate diffused from previous neighbor node along z-axis (from surface if border node) [W] | ** |
| 108 | *Psoil(6)* | Energy rate diffused from next neighbor node along z-axis (from surface if border node) [W] | ** |

| | | | |
|---|---|---|---|
| 109 | *Pint* | Energy rate of internal gains [W] | ** |

* averaged over node cluster

** integrated over node cluster and multiplied by number of modules

*Miscellaneous data for optional output :*

| **Type** | **Symbol** | **Definition and unit** |
|---|---|---|
| 201 | *PsurfTot* | Total inflowing energy rate through surfaces (over all modules) [kJ/hr] |
| 202 | *PwatTot* | Total energy loss of free water (over all modules) [kJ/hr] |
| 203 | *PfricTot* | Total frictional losses (over all modules) [kJ/hr] |
| 204 | *PintTot* | Total tube and soil capacitive gains (over all modules) [kJ/hr] |