Computer-aided process design for waste minimization

Konstadinos Abeliotis
New Jersey Institute of Technology

Follow this and additional works at: https://digitalcommons.njit.edu/dissertations

Part of the Chemical Engineering Commons

Recommended Citation
https://digitalcommons.njit.edu/dissertations/1107

This Dissertation is brought to you for free and open access by the Electronic Theses and Dissertations at Digital Commons @ NJIT. It has been accepted for inclusion in Dissertations by an authorized administrator of Digital Commons @ NJIT. For more information, please contact digitalcommons@njit.edu.
Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement.

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation.

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #” on the print dialog screen.
The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.
ABSTRACT

COMPUTER-AIDED PROCESS DESIGN FOR WASTE MINIMIZATION

by

Konstadinos Abeliotis

Problems related to process design with environmental considerations have been studied in this dissertation. A new heuristic design methodology has been formulated with the objective of minimizing the adverse environmental impact of chemical processes.

This hierarchical methodology starts with the definition and analysis of a process flowsheet followed by quantification of the process environmental loads. The quantification of environmental loads is an essential aspect of this methodology, because it serves as the comparative measure for environmental performance.

If the calculated environmental loads do not satisfy the environmental constraints (e.g., regulatory limits), waste minimization is investigated in the following order:

• Pollution prevention; process modifications aiming at pollution prevention are identified, prioritized, and evaluated.

• Pollution control; pollution control flowsheets are synthesized, analyzed, and evaluated based on the environmental loads.

The main focus of this research has been the partial implementation of this methodology on the computer to assist in the synthesis, analysis, and evaluation of pollution control flowsheets. Two separate programs have been developed:

• WasteMin Expert, a knowledge-based system which provides an organized framework for assisting the designer in the synthesis of pollution control flowsheets. It is an interactive rule-based expert system, which follows the task decomposition paradigm to accomplish the flowsheet synthesis task. The output of
WasteMin Expert provides a conceptual flowsheet in the form of an ordered list of key unit operations for pollution control. This list serves as a good starting point for the creation of detailed designs using a process simulator.

- EnviroCAD, a simulator developed for the analysis and evaluation of pollution control processes. EnviroCAD includes a variety of unit operation models used for waste treatment and disposal. EnviroCAD accounts for environmental loads on the basis of individual pollutants as well as aggregate environmental properties. Tracking the fate of individual pollutants facilitates retrofitting efforts to achieve the environmental objectives.

The methodology has been demonstrated in several examples, including the design of a wastewater treatment facility. The results of this case study are in good agreement with field data.
COMPUTER-AIDED PROCESS DESIGN FOR WASTE MINIMIZATION

Konstadinos Abeliotis

Dr. Demetri P. Petrides, Thesis Adviser
Assistant Professor of Chemical Engineering,
Chemistry, and Environmental Science, NJIT

Dr. Piero M. Armenante, Committee Member
Professor of Chemical Engineering,
Chemistry, and Environmental Science, NJIT

Dr. Basil C. Baltzis, Committee Member
Professor of Chemical Engineering,
Chemistry, and Environmental Science, NJIT

Dr. James Geller, Committee Member
Associate Professor of Computer and Information Sciences, NJIT

Dr. Dana E. Knox, Committee Member
Associate Professor of Chemical Engineering,
Chemistry, and Environmental Science, NJIT
BIOGRAPHICAL SKETCH

Author: Konstadinos Abeliotis

Degree: Doctor of Philosophy in Chemical Engineering

Date: October 1995

Undergraduate and Graduate Education:

• Doctor of Philosophy in Chemical Engineering, New Jersey Institute of Technology, Newark, NJ, 1995
• Diploma in Chemical Engineering, University of Patras, Patras, Greece, 1990

Major: Chemical Engineering

Publications:


Presentations:


This dissertation is dedicated to
my parents Georgio and Anna,
and my sisters Demetra and Vrisiida
ACKNOWLEDGEMENT

I would like to express my gratitude to my advisor, Dr. Demetri Petrides for his guidance and support throughout this research.

Special thanks to Professors Piero Armenante, Basil Baltzis, James Geller and Dana Knox for serving as members of my committee.

Financial support for this project by the Commission on Science and Technology of the State of New Jersey and a number of corporations (Eli Lilly, Pfizer, Sandoz, Schering Plough, and SmithKline Beecham) through the Emissions Reduction Research Center (ERRC) at NJIT, is gratefully acknowledged.

I am grateful to Dr. Vital Aelion for his numerous contributions to my graduate education and the time he spent discussing and reviewing this document. I would also like to thank Dr. Francesc Castells, Dr. Subir Mallick, and Dr. Ed Venkataramani for all their input and productive collaboration.

Special appreciation is extended to my fellow students Vinita Chhahira and Doug Hood, for their key contributions to the development of the expert system.

And finally, a special thank you to my family, and all my friends on both sides of the Atlantic, for their love and moral support throughout these years.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTRODUCTION</td>
</tr>
<tr>
<td>1.1 Thesis Objectives</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Thesis Organization</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>LITERATURE REVIEW</td>
</tr>
<tr>
<td>2.1 Waste Minimization</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Process Design</td>
<td>6</td>
</tr>
<tr>
<td>2.2.1 Process Synthesis</td>
<td>6</td>
</tr>
<tr>
<td>2.2.2 Process Analysis</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Process Design Methodologies for Waste Minimization</td>
<td>9</td>
</tr>
<tr>
<td>2.3.1 Hierarchical Decision Procedure</td>
<td>10</td>
</tr>
<tr>
<td>2.3.2 Knowledge-Based Artificial Intelligence Methods</td>
<td>11</td>
</tr>
<tr>
<td>2.3.3 Structural Optimization</td>
<td>11</td>
</tr>
<tr>
<td>2.3.4 Graph-Theoretic Approach</td>
<td>12</td>
</tr>
<tr>
<td>2.3.5 Mass Exchange Networks</td>
<td>12</td>
</tr>
<tr>
<td>2.4 Computer-Aided Tools for Waste Minimization</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>DESIGN METHODOLOGY FOR WASTE MINIMIZATION</td>
</tr>
<tr>
<td>3.1 Description of the Methodology</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>SYNTHESIS OF POLLUTION CONTROL FLOWSHEETS</td>
</tr>
<tr>
<td>4.1 Knowledge-Based Expert Systems</td>
<td>25</td>
</tr>
<tr>
<td>4.2 Synthesis Algorithm of WasteMin Expert</td>
<td>27</td>
</tr>
<tr>
<td>4.3 Technical Description of WasteMin Expert</td>
<td>36</td>
</tr>
<tr>
<td>4.3.1 Computer Implementation</td>
<td>36</td>
</tr>
<tr>
<td>4.3.2 Knowledge Base Structure</td>
<td>37</td>
</tr>
<tr>
<td>Chapter</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>5</td>
<td>ANALYSIS OF POLLUTION CONTROL PROCESSES ................................................. 39</td>
</tr>
<tr>
<td>5.1</td>
<td>Technical Description of EnviroCAD ............................................................ 40</td>
</tr>
<tr>
<td>5.1.1</td>
<td>The Unit Operation Structures ............................................................... 41</td>
</tr>
<tr>
<td>5.1.2</td>
<td>The Component Structure ......................................................................... 44</td>
</tr>
<tr>
<td>5.1.3</td>
<td>The Stream Structure ............................................................................. 45</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Flowsheet Representation in EnviroCAD .................................................... 46</td>
</tr>
<tr>
<td>5.2</td>
<td>User Interface of EnviroCAD .................................................................... 48</td>
</tr>
<tr>
<td>5.3</td>
<td>Economic Evaluation in EnviroCAD .............................................................. 48</td>
</tr>
<tr>
<td>6</td>
<td>QUANTIFICATION OF ENVIRONMENTAL LOADS IN ENVIROCAD ....................... 49</td>
</tr>
<tr>
<td>6.1</td>
<td>Pollution Inventories ................................................................................ 49</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Calculation of Process Emissions ............................................................... 51</td>
</tr>
<tr>
<td>6.2</td>
<td>Life Cycle Inventory Analysis .................................................................. 53</td>
</tr>
<tr>
<td>6.3</td>
<td>Environmental Stream Properties ............................................................... 59</td>
</tr>
<tr>
<td>7</td>
<td>UNIT OPERATIONS IN ENVIROCAD ............................................................... 64</td>
</tr>
<tr>
<td>7.1</td>
<td>Physical Wastewater Treatment Unit Operations ......................................... 64</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Equalization ............................................................................................. 65</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Discrete Particle Sedimentation ................................................................. 68</td>
</tr>
<tr>
<td>7.1.3</td>
<td>Dissolved-Air Flotation ............................................................................. 72</td>
</tr>
<tr>
<td>7.1.4</td>
<td>Granular Media Filtration ........................................................................... 76</td>
</tr>
<tr>
<td>7.1.5</td>
<td>Activated Carbon Adsorption .................................................................... 81</td>
</tr>
<tr>
<td>7.1.6</td>
<td>Air Stripping ............................................................................................. 85</td>
</tr>
<tr>
<td>7.2</td>
<td>Chemical Wastewater Treatment Unit Operations ......................................... 89</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS
(Continued)

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.1 Neutralization</td>
<td>90</td>
</tr>
<tr>
<td>7.3 Biological Wastewater Treatment Unit Operations</td>
<td>93</td>
</tr>
<tr>
<td>7.3.1 Aeration Basin</td>
<td>94</td>
</tr>
<tr>
<td>7.3.2 Trickling Filter</td>
<td>97</td>
</tr>
<tr>
<td>7.4 Sludge Treatment Unit Operations</td>
<td>101</td>
</tr>
<tr>
<td>7.4.1 Gravity Thickening</td>
<td>101</td>
</tr>
<tr>
<td>7.4.2 Anaerobic Digestion</td>
<td>105</td>
</tr>
<tr>
<td>7.4.3 Belt Filter Press</td>
<td>110</td>
</tr>
<tr>
<td>7.4.4 Heat Drying</td>
<td>112</td>
</tr>
<tr>
<td>7.4.4.1 Rotary Dryer</td>
<td>112</td>
</tr>
<tr>
<td>7.4.4.2 Spray Dryer</td>
<td>116</td>
</tr>
<tr>
<td>7.5 Air Pollution Control Unit Operations</td>
<td>120</td>
</tr>
<tr>
<td>7.5.1 Gas Absorption</td>
<td>120</td>
</tr>
<tr>
<td>7.5.2 Cyclone</td>
<td>124</td>
</tr>
<tr>
<td>7.5.3 Fabric Filtration</td>
<td>130</td>
</tr>
<tr>
<td>7.5.4 Electrostatic Precipitation</td>
<td>133</td>
</tr>
<tr>
<td>8 DEVELOPMENT OF A DESIGN CASE IN ENVIROCAD</td>
<td>137</td>
</tr>
<tr>
<td>8.1 Creating a Flowsheet</td>
<td>138</td>
</tr>
<tr>
<td>8.1.1 Select Unit Operations</td>
<td>138</td>
</tr>
<tr>
<td>8.1.2 Draw Streams</td>
<td>139</td>
</tr>
<tr>
<td>8.2 Initializing a Flowsheet</td>
<td>139</td>
</tr>
<tr>
<td>8.2.1 Initialize Components</td>
<td>140</td>
</tr>
<tr>
<td>8.2.2 Initialize Feed Streams</td>
<td>140</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

(Continued)

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.2.3</td>
<td>Initialize Unit Operations</td>
</tr>
<tr>
<td>8.2.4</td>
<td>Initialize Cost Data</td>
</tr>
<tr>
<td>8.3</td>
<td>Analyzing a Flowsheet</td>
</tr>
<tr>
<td>8.3.1</td>
<td>Solve M&amp;E Balances</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Generate Stream Report</td>
</tr>
<tr>
<td>8.3.3</td>
<td>Perform Economic Analysis</td>
</tr>
<tr>
<td>8.4</td>
<td>Process Emissions</td>
</tr>
<tr>
<td>8.5</td>
<td>Life Cycle Inventory Analysis</td>
</tr>
<tr>
<td>9</td>
<td>EXAMPLE: DESIGN OF A POLLUTION CONTROL FACILITY</td>
</tr>
<tr>
<td>9.1</td>
<td>Synthesis of Wastewater Treatment Facility</td>
</tr>
<tr>
<td>9.2</td>
<td>Analysis of Wastewater Treatment Facility</td>
</tr>
<tr>
<td>9.3</td>
<td>Process Modifications</td>
</tr>
<tr>
<td>10</td>
<td>CONTRIBUTIONS AND RECOMMENDATIONS</td>
</tr>
<tr>
<td>10.1</td>
<td>Research Results</td>
</tr>
<tr>
<td>10.2</td>
<td>Recommendations for Future Work</td>
</tr>
<tr>
<td>APPENDIX I:</td>
<td>UNIT OPERATION DATA STRUCTURES IN ENVIROCAD</td>
</tr>
<tr>
<td>APPENDIX II:</td>
<td>UNIT OPERATION MODELS IN ENVIROCAD</td>
</tr>
<tr>
<td>APPENDIX III:</td>
<td>PROCESS EMISSIONS IN ENVIROCAD</td>
</tr>
<tr>
<td>APPENDIX IV:</td>
<td>LIFE CYCLE INVENTORY ANALYSIS IN ENVIROCAD</td>
</tr>
<tr>
<td>APPENDIX V:</td>
<td>ENVIRONMENTAL STREAM PROPERTIES IN ENVIROCAD</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>317</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Environmental Stream Properties in EnviroCAD.</td>
<td>60</td>
</tr>
<tr>
<td>7.1</td>
<td>Discrete Particle Settling Regimes</td>
<td>71</td>
</tr>
<tr>
<td>9.1</td>
<td>Composition of Plant Influent.</td>
<td>149</td>
</tr>
<tr>
<td>9.2</td>
<td>Questions asked by WasteMin Expert.</td>
<td>150</td>
</tr>
<tr>
<td>9.3</td>
<td>User Input and WasteMin Expert Responses</td>
<td>151</td>
</tr>
<tr>
<td>9.4</td>
<td>Removal Percentages in the Aeration Basin</td>
<td>154</td>
</tr>
<tr>
<td>9.5</td>
<td>Overall Performance Based on Environmental Stream Properties (kg/day)</td>
<td>155</td>
</tr>
<tr>
<td>9.6</td>
<td>Aeration Basin Gas Outlet (stream S106)</td>
<td>156</td>
</tr>
<tr>
<td>9.7</td>
<td>Secondary Clarifier Overflow Stream (S126)</td>
<td>157</td>
</tr>
<tr>
<td>9.8</td>
<td>Performance Comparison Between the Original and the Modified Flowsheet</td>
<td>158</td>
</tr>
<tr>
<td>9.9</td>
<td>Capital and Operating Costs of the Two Alternative Flowsheets</td>
<td>160</td>
</tr>
<tr>
<td>9.10a</td>
<td>Major Equipment Specification and FOB Cost (1995 Prices)</td>
<td>162</td>
</tr>
<tr>
<td>9.10b</td>
<td>Fixed Capital Estimate Summary (1995 Prices)</td>
<td>163</td>
</tr>
<tr>
<td>9.10c</td>
<td>Annual Operating Cost (1995 Prices)</td>
<td>164</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The Waste Management Hierarchy</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Main Components of a Process Simulator</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>Decision Diagram of the Design Methodology</td>
<td>18</td>
</tr>
<tr>
<td>4.1</td>
<td>Abstract Decision Diagram of WasteMin Expert</td>
<td>27</td>
</tr>
<tr>
<td>4.2</td>
<td>Solvent Recovery Decision Diagram in WasteMin Expert</td>
<td>28</td>
</tr>
<tr>
<td>4.3</td>
<td>Pre-treatment Decision Diagram in WasteMin Expert</td>
<td>29</td>
</tr>
<tr>
<td>4.4</td>
<td>Primary Treatment Decision Diagram in WasteMin Expert</td>
<td>30</td>
</tr>
<tr>
<td>4.5</td>
<td>Secondary Treatment Decision Diagram in WasteMin Expert</td>
<td>31</td>
</tr>
<tr>
<td>4.6</td>
<td>Tertiary Treatment Decision Diagram in WasteMin Expert</td>
<td>33</td>
</tr>
<tr>
<td>4.7a</td>
<td>Sludge Treatment Decision Diagram in WasteMin Expert</td>
<td>34</td>
</tr>
<tr>
<td>4.7b</td>
<td>Sludge Treatment Decision Diagram in WasteMin Expert</td>
<td>35</td>
</tr>
<tr>
<td>4.8</td>
<td>Sludge Disposal Decision Diagram in WasteMin Expert</td>
<td>36</td>
</tr>
<tr>
<td>5.1</td>
<td>The Unit Operation Data Structures</td>
<td>42</td>
</tr>
<tr>
<td>5.2</td>
<td>The Hierarchy of Unit Operation Structures in EnviroCAD</td>
<td>43</td>
</tr>
<tr>
<td>5.3</td>
<td>The Component Data Structure</td>
<td>44</td>
</tr>
<tr>
<td>5.4</td>
<td>The Stream Data Structure</td>
<td>45</td>
</tr>
<tr>
<td>5.5</td>
<td>Flowsheet Structure in EnviroCAD</td>
<td>47</td>
</tr>
<tr>
<td>6.1</td>
<td>Representation of Mass Flow in a Manufacturing Process</td>
<td>50</td>
</tr>
<tr>
<td>6.2</td>
<td>Technical Framework for Life Cycle Analysis</td>
<td>54</td>
</tr>
<tr>
<td>6.3</td>
<td>Generic Process Representation</td>
<td>56</td>
</tr>
<tr>
<td>6.4</td>
<td>Implementation of Mass Eco-vector in EnviroCAD</td>
<td>57</td>
</tr>
<tr>
<td>6.5</td>
<td>Implementation of Energy Eco-vector in EnviroCAD</td>
<td>58</td>
</tr>
<tr>
<td>6.6</td>
<td>Output Dialog Window for Environmental Stream Properties</td>
<td>59</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>6.7</td>
<td>Distribution of Nitrogen Content in EnviroCAD</td>
<td>61</td>
</tr>
<tr>
<td>6.8</td>
<td>Distribution of Solids Content in EnviroCAD</td>
<td>61</td>
</tr>
<tr>
<td>6.9</td>
<td>Input Dialog Window for Contribution Factors</td>
<td>62</td>
</tr>
<tr>
<td>7.1</td>
<td>Solution Algorithm for the Equalization Model</td>
<td>68</td>
</tr>
<tr>
<td>7.2</td>
<td>Solution Algorithm for the Discrete Particle Sedimentation Model</td>
<td>72</td>
</tr>
<tr>
<td>7.3</td>
<td>Solution Algorithm for the Dissolved-Air Flotation Model</td>
<td>75</td>
</tr>
<tr>
<td>7.4</td>
<td>Solution Algorithm for the Granular Media Filtration Model</td>
<td>80</td>
</tr>
<tr>
<td>7.5</td>
<td>Solution Algorithm for the Activated Carbon Adsorption Model</td>
<td>84</td>
</tr>
<tr>
<td>7.6</td>
<td>Solution Algorithm for the Air Stripping Model</td>
<td>89</td>
</tr>
<tr>
<td>7.7</td>
<td>Solution Algorithm for the Neutralization Model</td>
<td>93</td>
</tr>
<tr>
<td>7.8</td>
<td>Solution Algorithm for the Aeration Basin Model</td>
<td>97</td>
</tr>
<tr>
<td>7.9</td>
<td>Solution Algorithm for the Trickling Filter Model</td>
<td>101</td>
</tr>
<tr>
<td>7.10</td>
<td>Solution Algorithm for the Gravity Thickening Model</td>
<td>105</td>
</tr>
<tr>
<td>7.11</td>
<td>Solution Algorithm for the Anaerobic Digestion Model</td>
<td>109</td>
</tr>
<tr>
<td>7.12</td>
<td>Solution Algorithm for the Belt Filter Press Model</td>
<td>112</td>
</tr>
<tr>
<td>7.13</td>
<td>Solution Algorithm for the Rotary Dryer Model</td>
<td>116</td>
</tr>
<tr>
<td>7.14</td>
<td>Solution Algorithm for the Spray Dryer Model</td>
<td>119</td>
</tr>
<tr>
<td>7.15</td>
<td>Solution Algorithm for the Absorption Model</td>
<td>124</td>
</tr>
<tr>
<td>7.16</td>
<td>Solution Algorithm for the Cyclone Model</td>
<td>129</td>
</tr>
<tr>
<td>7.17</td>
<td>Solution Algorithm for the Fabric Filtration Model</td>
<td>133</td>
</tr>
<tr>
<td>7.18</td>
<td>Solution Algorithm for the Electrostatic Precipitation Model</td>
<td>136</td>
</tr>
<tr>
<td>8.1</td>
<td>The Unit Operations Menu</td>
<td>138</td>
</tr>
<tr>
<td>8.2</td>
<td>Stream Representation in EnviroCAD</td>
<td>139</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES
(Continued)

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3</td>
<td>The Tasks Menu</td>
</tr>
<tr>
<td>8.4</td>
<td>First Dialog Window for Initialization of Components</td>
</tr>
<tr>
<td>8.5</td>
<td>Dialog Window for Initialization of Feed Streams</td>
</tr>
<tr>
<td>8.6</td>
<td>First Dialog Window of the Clarifier Model</td>
</tr>
<tr>
<td>8.7</td>
<td>The Emissions Menu</td>
</tr>
<tr>
<td>8.8a</td>
<td>Specification of Components Considered in Emissions Calculations</td>
</tr>
<tr>
<td>8.8b</td>
<td>Selection of Unit Operations Considered for Emissions Calculations</td>
</tr>
<tr>
<td>8.8c</td>
<td>Dialog Window for the Selection of Tasks</td>
</tr>
<tr>
<td>8.8d</td>
<td>Dialog Window for the Initialization of the Charging Task</td>
</tr>
<tr>
<td>8.9</td>
<td>The Life Cycle Inventory Analysis Menu</td>
</tr>
<tr>
<td>8.10</td>
<td>Characterization of Output Streams</td>
</tr>
<tr>
<td>9.1</td>
<td>Conceptual Flowsheet Generated by WasteMin Expert</td>
</tr>
<tr>
<td>9.2</td>
<td>Wastewater Treatment Facility</td>
</tr>
<tr>
<td>9.3</td>
<td>Modified Wastewater Treatment Facility</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

The generation of wastes in any industrial activity is inevitable. The release of waste to the environment contributes to pollution, which in turn poses a threat to human health and the ecological balance. The problem of minimizing the generation of waste and controlling the release of pollutants is old, but there is a renewed interest in addressing it, mainly because of:

• Increasingly stricter environmental regulations,
• Rapidly rising waste treatment and disposal costs, and
• Increased public awareness for the environment.

The solution to this problem is waste minimization. Waste minimization can be implemented in two levels:

• Pollution prevention, i.e., the series of proactive steps taken to lessen or eliminate the generation of waste, and
• Pollution control, i.e., the application of end-of-pipe measures to reduce the release of pollutants to the environment.

Although pollution prevention is typically the preferred waste minimization alternative, it is not always feasible. Frequently, the combination of pollution prevention and pollution control is the practical approach.

Waste generation is closely related to process design. For example, a better designed reaction sequence in a process results in better utilization of raw materials, solvents, and catalysts which in turn results in higher product yield and lower waste generation. Similarly, an energy efficient plan, results in reduced NOx, SOx, and CO2 releases to
the environment. These observations indicate that waste minimization can be achieved by incorporating environmental considerations and constraints in process design.

The design of processes involves the thoughtful integration of individual process components. Design methodologies offer a structured approach for achieving this integration by using heuristic and/or mathematical knowledge. The successful combination of these two types of knowledge is part of any successful design methodology.

The development of better designs is the main benefit resulting from a powerful design methodology. Also, such a methodology offers a teachable body of knowledge, which can be used to formally educate students and engineers and empower them to produce better designs.

1.1 Thesis Objectives

The focus of this thesis is to:

- Formulate a methodology which incorporates principles of waste minimization in the design of chemical processes.
- Develop computer tools that facilitate the automation of parts of the methodology.
- Demonstrate the application of the computer tools for the design of pollution control processes.

1.2 Thesis Organization

Chapter 2 reviews methodologies and computer tools used for environmentally-oriented process design. In Chapter 3, the design methodology developed as part of this work is presented. Chapter 4 describes WasteMin Expert, an expert system developed to assist in the synthesis of pollution control processes.
Chapters 5-8 address various aspects of EnviroCAD, the process simulator developed for the analysis of pollution control processes. Chapter 5 presents EnviroCAD from the developer's technical point of view. Chapter 6 describes the features of the program which allow the quantification of the environmental loads, while Chapter 7 presents the description of the unit operation models contained in EnviroCAD. Chapter 8 outlines the development of a design case in EnviroCAD.

Chapter 9 demonstrates the application of WasteMin Expert and EnviroCAD in the design of a wastewater treatment facility. Chapter 10 presents results, conclusions, and recommendations for future work. Finally, the five appendices contain the source code of EnviroCAD.
CHAPTER 2

LITERATURE REVIEW

2.1 Waste Minimization

Waste minimization is defined by the U. S. Environmental Protection Agency (EPA) as the reduction, to the extent feasible, of hazardous waste that is generated or subsequently treated, sorted or disposed. It includes any source reduction or recycling activity undertaken by a generator that results in either (i) the reduction of total volume or quantity of hazardous waste, or (ii) the reduction of toxicity of hazardous waste, or both, so long as such reduction is consistent with the goal of minimizing recent and future threats to human health and the environment (U. S. EPA, 1986).

The Pollution Prevention Act of 1990 introduced the waste minimization hierarchy (Figure 2.1). Reduction of pollution at the source is at the top of this hierarchy. In-process recycling is at the next level. On-site and off-site recycling, treatment and disposal follow at subsequent levels. Source reduction and in-process recycling are the only waste minimization options that qualify as pollution prevention, according to the EPA (U. S. EPA, 1990).

In addition to the Pollution Prevention Act, two more key federal programs have been implemented in the United States so far to address pollution prevention (Doerr, 1993): the EPA's 33/50 Voluntary Reduction Program and the Clean Air Act Amendments' Early Reduction Program. Also, half of the states have pollution prevention programs enacted or proposed. Among them, New Jersey has a comprehensive pollution prevention program (Allyn, 1993).

Pollution prevention is the preferred waste minimization option because it has many advantages:
Figure 2.1 The Waste Management Hierarchy

First, it results in lower operating costs due to better utilization of raw materials and energy, and reduced waste treatment and disposal costs. The EPA estimates that hazardous waste treatment and disposal costs have risen as much as 300% over the past decade (Clearwater and Scanlon, 1991). Today, disposal costs vary anywhere from $100 to more than $1,000 on a per ton basis (Lederman and Weaver, 1991).

Secondly, pollution prevention facilitates the compliance with the environmental laws and regulations. For example, the Pollution Prevention Act of 1990 anticipates penalties up to $25,000 per day for each violation. Furthermore, mandatory jail time has become an increasingly common reality for environmental crimes (Clearwater and Scanlon, 1991). Pollution prevention also helps avoiding both civil and criminal liabilities related to today's disposal sites which turn out to be sources of contamination
in the future. Future liability costs are estimated in the order of $300-$500 per ton of waste (Jacobs, 1991).

Finally, pollution prevention enhances the public image of an organization, which is important to consumers and suppliers ("Green Economy"), the surrounding community, the regulators and political decision makers, and the public in general.

2.2 Process Design

Process design is the creative activity that leads from the identification of a need to an end process that satisfies that need and it is subject to economic, operational, safety, environmental, and other constraints. The relative importance of these constraints has varied over the years. Nowadays, the burden on the environmental resources and the strict regulations have increased interest in minimizing the environmental impact of industrial processes. Hence, environmental constraints in product and process design have become important.

Design is to a large extent an information processing activity, because it depends on numerous areas of knowledge such as reaction kinetics, thermodynamics, material science, etc. Process design consists of two main sub-activities, process synthesis and process analysis.

2.2.1 Process Synthesis

Process synthesis is the creation of an operational system, composed of individual processing units interacting with each other, in such a way that:

- certain economic and operational objectives are optimized,
- production specifications are met, and
- operating restrictions determined by the processing units, safety codes, environmental regulations, etc. are satisfied (Stephanopoulos, 1981).
Process synthesis is an open-ended combinatorial problem due to the large number of processing units available and their resulting topologies.

Rudd (1968) and his students (Powers, 1972; Siirola and Rudd, 1971) at the University of Wisconsin-Madison pioneered the research in automation of synthesis of entire flowsheets. Since then, process synthesis has emerged as a major research topic in chemical engineering. Various approaches and methodologies have been used to address the problem with variable degrees of success. Extensive review articles about process synthesis can be found in the literature (Westerberg, 1989; Umeda, 1983; Nishida et al., 1981; Hlavacek, 1978). Synthesis methodologies for waste minimization are presented in section 2.3.

2.2.2 Process Analysis

Process analysis is the examination and evaluation of the alternative solutions generated by process synthesis. In order to reduce costly and time consuming experiments, process analysis makes extensive use of process simulation. Simulation is the use of computer aids to perform mass and energy balancing, sizing and costing calculations for a process flowsheet.

The use of simulation has expanded rapidly due to the development of mathematical algorithms for the solution of the systems of algebraic and differential equations needed to describe a flowsheet. Also, the exponential growth in power of personal computers has drastically reduced the cost of computer time and has moved process simulators from mainframes to personal computers. Several software packages have been developed mainly for simulation of continuous chemical processes and are by now mature. Figure 2.2 shows the main components of a process simulator. The heart of any process simulator is the ability to model the performance of unit operations that compose a flowsheet.
Two main approaches are used in the development of a simulation system (Westerberg et al., 1979): (i) sequential modular, and (ii) equation oriented.

In the *sequential modular* approach, the basic philosophy is that each unit model calculates all output stream values, given values for all input streams and for all equipment parameters. Also fundamental to this approach is that recycle stream variables are estimated via iterative calculations, until flowsheet convergence is achieved. This approach is natural to the user because information flow follows flow of material in the streams. It is also, easier to implement and debug, compared to the other approaches. However, it is not efficient in handling complex processes with a large number of nested recycle loops. Also, it is not well suited for the solution of optimization problems because the entire flowsheet may have to be solved iteratively a large number of times before the optimum solution is found.
In the equation oriented approach, the entire set of equations that describe a flowsheet are solved simultaneously. The equations involved include user supplied specifications, mass and energy balances, reaction equations, equations connecting the units within the flowsheet, physical properties estimation correlations, etc. Since the equations are solved simultaneously, there is no need for iterative procedures to solve the recycle loops. Since the simultaneous equations can be used as constraints in a generalized programming problem, this approach is well suited for process optimization. However, this approach has a number of problems. It is computationally more demanding, because it requires solution of large systems of equations. Frequently it fails to converge the large systems of equations for real world problems. Another problem is that the flexibility associated with the equation oriented approach often leads to inconsistent specifications that are hard to identify and correct.

The major benefit resulting from process simulation is that it allows the investigation of the effect of various parameters on the operation of a system and the rapid evaluation of different process alternatives. Simulation also offers a reasonable way of extrapolating performance and scaling up processes. On the other hand, successful simulations require good process data, knowledge of process mechanisms and availability of numerical tools to solve the systems of equations that arise from modeling. Also, the simplifying assumptions made during the modeling process may have an effect on the simulation results.

2.3 Process Design Methodologies for Waste Minimization

The production of waste from a chemical process is a function of the process design and the manner in which the process is operated (Manousiouthakis and Allen, 1995; Hollod and McCartney, 1988). For example, a better designed reaction sequence in a process results in better utilization of raw materials, solvents, and catalysts which in turn results
in higher product yield, less by-products, and lower waste generation. Also, an energy efficient plant, in the environmental context, results in reduced NO\textsubscript{x}, SO\textsubscript{x}, and CO\textsubscript{2} releases to the environment. Therefore, waste minimization is closely related to process design.

As mentioned earlier in this chapter, many approaches have been used to address process synthesis. They can be categorized, based on their underlying principles, as:

- heuristic,
- mathematical, or
- "thermodynamic."

Combinations of the above categories are also used. Some approaches deal with the synthesis of entire flowsheets, while others address synthesis of specific subsystems such as separation sequences. Synthesis approaches with application to waste minimization are presented in the following sections.

2.3.1 Hierarchical Decision Procedure

Douglas (1992) adapted his hierarchical decision procedure for process synthesis (Douglas, 1985), to waste minimization problems. It is a general grassroots design methodology which is based on the decomposition of a problem into smaller problems in a structured and organized manner. Process design is generated through a series of hierarchical decisions which provide successive levels of detail. These decisions are based on engineering judgment and experience. Applications of this methodology to retrofitting industrial waste minimization design problems appeared in the literature recently (Fonyo et al., 1994; Rossiter et al., 1993).

This methodology is of great teaching value because it provides a structured framework of thinking. On the other hand, it does not guarantee finding the optimum
design and cannot be easily implemented on the computer because it deals with synthesis of entire flowsheets, a problem that has not found a satisfactory solution yet.

2.3.2 Knowledge-Based Artificial Intelligence Methods
The major advantage of knowledge-based Artificial Intelligence (AI) methods is that they can control rapidly expanding solution spaces, such as those involved in synthesis problems, using heuristic knowledge. Heuristics provide useful insights for the development of good designs because they are usually based on accumulated actual field experience. Articles describing design heuristics aiming at waste minimization, resulting from the industrial practice of large chemical firms, have appeared in the literature (Rossiter et al., 1993; Haseltine, 1992; Benforado et al., 1991; Nelson, 1990).

Recent approaches in process design, use hybrid methods which utilize heuristic knowledge in addition to other AI techniques (Kirkwood et al., 1988; Lu and Motard, 1985) for the synthesis of total flowsheets. With respect to waste minimization, Rossman (1989) uses the artificial intelligence paradigms of algorithmic state space search guided by knowledge-based heuristics to synthesize a waste treatment flowsheet.

2.3.3 Structural Optimization
The main idea in this approach is to formulate the synthesis of a flowsheet as an optimization problem (Grossmann, 1985; Umeda et al., 1972). This algorithmic approach has been used by several authors (Camara, 1984; Rossman, 1980; Bush and Silveston, 1978) for synthesis of waste treatment systems. In this approach, many alternatives are embedded into a superstructure, in which all possible connections among process units are included. The superstructure is then optimized by means of an appropriate optimization technique, with respect to one or more design objectives. The
application of structural parameter optimization to waste treatment systems synthesis is often limited to a small number of units and operating parameters.

Ciric and Jia (1994) modeled the chemical process synthesis problem as a multiobjective optimization problem that seeks to simultaneously maximize profit and minimize waste. Recently, Pistikopoulos et al. (1994) proposed a design methodology for Minimum Environmental Impact (MEI) which embeds principles of Life Cycle Analysis (Society of Environmental Toxicology and Chemistry, 1993) within a structural optimization framework. In this methodology, the optimization of the superstructure is done with respect to environmental indices which measure air pollution, water pollution, solid wastes, global warming, photochemical oxidation, and stratospheric ozone depletion.

2.3.4 Graph-Theoretic Approach
This approach can be used for the generation of the superstructures needed for the application of structural optimization. It is an algorithmic-oriented approach for process synthesis which relies heavily on the graph theory and combinatorial techniques (Friedler et al., 1992). At first, an algorithm generates the maximal structure of the process synthesis problem in which all feasible structures are embedded. Individual solution-structures can be extracted from the maximal structure subject to various technical, economic, environmental, and societal constraints. The application of this theory in a waste minimization problem is demonstrated in Friedler et al. (1990).

2.3.5 Mass Exchange Networks
The synthesis of mass exchange networks (El-Halwagi and Manousiouthakis, 1989) is based on the methodology of Pinch Analysis (Linnhoff and Hindmarch, 1983) developed for the synthesis of heat exchange networks. The objective of a mass
exchange network is the preferential transfer of certain species (e.g., pollutants) from a set of rich (with respect to the transferred species) streams to a set of lean streams, in a manner similar to the transfer of heat from a set of hot streams to a set of cold streams.

This approach is based on a very powerful concept: The best achievable performance of a system (e.g., minimum amount of lean stream, minimum number of separating units) is determined before the actual design of the system is known, by identifying the thermodynamic bottlenecks (pinch points) that limit the extent of mass transfer. The best achievable performance is targeted, and a design is generated that achieves this performance. This approach is therefore very likely to produce a design which is close to the optimum.

The mass exchange networks were originally used for the design of separation systems. Recently, they have been applied to waste minimization problems (Wang and Smith, 1994; El-Halwagi, 1992; El-Halwagi et al., 1992).

2.4 Computer-Aided Tools for Waste Minimization

In recent years, there has been great interest and activity in academia, government and the private sector for the development of computer tools with direct application in various sectors of the environmental field (Simpson, 1994; Johnson, 1993). The vast majority of these tools deal with issues of pollution monitoring and reporting, and regulatory compliance. Very few of these tools address waste minimization. The following paragraphs describe analytic and synthetic tools aiming at waste minimization.

A very ambitious system is at the very early stages of development at the Center for Clean Industrial and Treatment Technologies at the Michigan Technological University. The Clean Process Advisory System™ (Hertz et al., 1994) is a computer-based pollution prevention process and product design system. It will be a collection of various software tools, each addressing specific clean technology information needs.
BatchDesign-Kit (Linninger et al., 1994) is a computer tool which implements a pollution prevention design methodology for batch processes. It has two components, one used for process synthesis and the other for analysis and assessment of the generated design. The synthetic component of BatchDesign-Kit, based on the hierarchical decision procedure (see section 2.3.1) and knowledge-based expert systems, generates ideal Zero Avoidable Pollution (ZAP) processes. Those processes are then analyzed by the analytic component until the optimum trade-off between cost and generation of wastes is achieved. This analysis eventually leads to practical Minimum Avoidable Pollution (MAP) batch processes.

A prototype simulation program dealing with wastewater treatment plant design was developed at the University of Illinois at Urbana-Champaign by Tang et al. (1987). The base model is for an activated sludge secondary wastewater treatment plant. It can also simulate certain units for sludge processing (gravity thickening, anaerobic digestion, vacuum filtration) and disposal (landfilling). The prototype features a simple user interface and it can be used in an analysis mode or an optimization mode. Its major limitation is that it only includes a small number of unit operations and therefore it is not possible to simulate an entire wastewater treatment plant. Also, the material balances are executed based on lumped environmental stream properties (e.g., BOD, COD, etc.) and as a result it cannot be used to track the fate of individual hazardous chemicals in integrated waste treatment facilities.

Several process simulation companies have developed tools for waste minimization. ENPRO (Simulation Sciences Inc., 1992), in addition to models for primary and secondary wastewater treatment, includes models for absorption, stripping, and activated carbon adsorption. ENPRO lacks economic evaluation capabilities, a feature essential for comparing process alternatives.
ESP (Environmental Simulation Program) is a set of modeling and simulation modules for environmental applications (OLI Systems Inc., 1993). The main strength of ESP is in modeling reaction and equilibrium phenomena in aqueous systems involving molecular as well as ionic species. ESP, like ENPRO, lacks economic evaluation capabilities.

In addition to simulation software, many expert systems have been developed to identify and control environmental problems. Two review articles (Greathouse et al., 1989; Hushon, 1987) provide descriptive lists of expert systems used for this purpose.

With respect to the application of expert systems in waste minimization, Fan and his students at Kansas State University have done considerable work (Huang and Fan, 1993). For example, MIN-CYANIDE (Huang et al., 1991) is an expert system for the minimization of cyanide waste generation in electroplating plants. It evaluates different plant operation alternatives towards waste minimization and identifies the most effective among them. The knowledge base of the system contains detailed cyanide minimization strategies while its database includes the process data and constraints. This expert system can handle ambiguous or imprecise information since it utilizes elements of fuzzy logic for knowledge representation.

Design methodologies can be complemented by the quantification of environmental loads, which provides a common ground for comparison of various processes. In this thesis, a design methodology has been developed which makes extensive use of quantification of environmental loads. Emphasis has also been placed in the partial computer implementation of this methodology in order to reduce the time required for the examination of alternative designs and produce better results. This new methodology is presented in the next chapter.
CHAPTER 3

DESIGN METHODOLOGY FOR WASTE MINIMIZATION

Process design is an opportunistic activity. Based on the design goals and constraints, a search space is defined and then an algorithm is employed to find a solution to the design problem within the search space. This solution should satisfy the goals and meet the constraints of the problem. If required by the designer (or the problem), this solution may sometimes have some special properties, such as optimality with respect to a pre-defined evaluation function.

Process design integrates knowledge from many different areas (e.g., reaction kinetics, thermodynamics, separation processes, material science, etc.), and it is subject to a variety of constraints (economic, environmental, technological, safety, etc.), whose relative importances vary. Environmental considerations in process design have become very important recently, as a result of increased government regulations and public awareness. The objective of these environmental considerations is to minimize the adverse effects of the industrial activities to humans and to the surrounding environment.

This chapter presents a new methodology which incorporates environmental considerations in the retrofit design of chemical processes. The output of this methodology is a conceptual chemical process which has minimal environmental impact. This methodology aims at offering a structured way of searching the design space and exploring different design alternatives. It is a design methodology that combines both synthetic and analytic components. Parts of the methodology have been automated on the computer.
3.1 Description of the Methodology

There are two common characteristics in many design methodologies:

- All the activities involved in the design process are frequently placed in a hierarchy.
- There are activity blocks that represent logical connections of some of the activities to certain goals.

The methodology presented here follows the hierarchical approach. It consists of two major activity blocks: the pollution prevention block, and the pollution control block. The decision diagram of the methodology is shown in Figure 3.1. In this figure, activities are represented by rectangles, while decision points are shown by ovals. The methodology is described in detail in the following paragraphs.

1. Specification of process flowsheet

The input to the methodology is a flowsheet of a chemical/biochemical plant. The flowsheet may be either the representation of an existing chemical process or a new process under development. Flowsheet specifications include:

- The unit operations of the process.
- The stream and recycle structure.
- The operating conditions and parameters of the unit operations.
- Supporting information, such as physical and chemical properties of the components, regulatory information, etc.

2. Analysis of process flowsheet

Based on flowsheet specifications, the material and energy balances are solved and the unit operations of the flowsheet are sized using simulation.
Figure 3.1 Decision Diagram of the Design Methodology

3. Quantification of environmental loads

Based on the simulation results, the environmental loads of the waste streams of the process are quantified. In order to provide a common ground for comparison of various
processes, quantifiable entities which can be estimated for every process need to be used. Such entities include:

- Lumped environmental stream properties such as total organic carbon (TOC) and biochemical oxygen demand (BOD). These properties provide useful insights into desirable stream characteristics, such as potential biodegradability. Also very often the regulatory discharge limits to liquid bodies (e.g., rivers and lakes) are set with respect to these lumped properties.

- Total amounts of regulated priority pollutants (e.g., 33/50 EPA program), which are present in the waste streams. Most regulatory limits are based on the total amount of releases for each individual regulated chemical.

- Various pollution indices. For example, one index could be defined as the ratio of the amount of waste generated over the amount of product produced. Depending on the relevant application, multiple indices can be defined based on different process inputs and/or outputs.

- Eco-vectors, derived from the Life Cycle Analysis methodology (Society of Environmental Toxicology and Chemistry, 1993), which is the basis for thorough accounting of environmental loads because it considers the environmental loads resulting from a process as well as from raw materials and energy input.

The quantification of the environmental load of waste streams supports also the ranking of the streams and the potential creation of a priority list based on one or more of the above quantities.

4. Is environmental load within acceptable limits?

Once the environmental load for each waste stream is quantified, a decision has to be made by the designer on whether or not this environmental load satisfies some pre-
specified criteria (constraints). These criteria may be based on one or more of the following reasonings:

- Current federal, state or local regulations.
- Anticipated future regulations.
- Total waste generation levels.
- Levels of lumped environmental stream properties.
- Company policies and corporate goals.
- Engineering judgment.

If the environmental load for each waste stream as well as the total process load are within acceptable limits, then the process can be considered environmentally benign and there is no need for waste minimization.

If process discharges exceed regulatory limits or large amounts of waste are generated, then the process is not environmentally acceptable and additional actions need to be taken to minimize waste. Waste minimization efforts should be focused primarily on streams with large environmental loads.

5. Are there any pollution prevention opportunities?

Waste minimization efforts should start with pollution prevention. From the process design point of view, pollution prevention can be implemented via process modifications. Based on the quantification of environmental loads (step 3), the search for possible process modifications should start from the unit operations that generate the waste streams with the highest environmental loads. The successful identification of process opportunities depends largely on the experience, creativity, and intuition of the human designer. Usually, a heuristic approach posing questions such as the following, may prove to be very helpful.
• Do alternative processes exist that produce the same product but reduce the environmental burden?
• Is it feasible to substitute process raw materials and/or solvents in order to eliminate one or more waste streams?
• Is it possible to increase reaction yield or selectivity by changing process operating conditions in order to avoid the generation of unwanted by-products or waste?
• Is it possible to recycle in-process material that ends up in a waste stream?
• Is it feasible to improve the separations sequence in order to reduce waste generation?
• Is it possible to use the waste generated from this process as a raw material for another process?

If several pollution prevention opportunities can be identified, they should be prioritized (step 6). If there are no process modifications that can be made towards pollution prevention, waste minimization is examined via pollution control (step 9).

6. Prioritization of pollution prevention options

If several opportunities for pollution prevention have been identified, they need to be evaluated and ranked so that the more promising ones can be pursued first. The evaluation of those options may be based on one or more engineering criteria. Potential ranking criteria include:

• Regulatory constraints.
• Capital and/or operating cost.
• Costs per unit of waste reduced.
• Anticipated reduction in waste generation.
• Raw materials utilization.
If the ranking is based on one criterion it is straightforward; if multi-criteria ranking is needed, a utility function or a decision support methodology, such as the Analytic Hierarchy Process (Saaty, 1986), may be used.

7. Is pollution prevention favorable over pollution control?

Once the pollution prevention options have been prioritized, the strategic decision that must be made is whether or not pollution prevention is preferable over pollution control. Since pollution prevention in process design is implemented via process modifications, the major criterion should be the economic potential of the modified process. If financial indices, such as the Net Present Value or the Internal Rate of Return, of the modified process are favorable, pollution prevention should be actively pursued.

The difficulty arises in the estimation of these indices. Based on traditional economic measures, pollution control frequently appears more attractive. However, if hidden environmental costs (permit requirements, monitoring), future liabilities (penalties, fines) and less tangible costs (corporate image) of waste generation are taken into account, pollution prevention options may be more competitive (Berger et al., 1994; Karam et al., 1988). The Total Cost Assessment methodology (White, 1993) offers a way for a complete cost accounting. However, a complete cost accounting is a very difficult task because there is a great uncertainty about the hidden environmental costs. One has to rely on financial data derived from similar operations in the past, or acceptable average values.

If a decision favoring pollution prevention is made, the designer has to analyze and examine carefully the most prominent (or all) of the proposed process modifications (step 8). Otherwise, pollution control should be considered (step 9).
8. Specification of pollution prevention flowsheet

For each of the proposed pollution prevention alternatives, a new flowsheet has to be specified and analyzed. This new flowsheet is the result of the proposed modifications imposed on the initial process. The pollution prevention design loop starts.

Each new flowsheet is analyzed and the environmental load of each waste stream is evaluated. If an acceptable process is found within the proposed options, the loop ends and a successful pollution prevention design has been achieved. If all the suggested pollution prevention alternatives fail, the loop ends and waste minimization is examined using pollution control.

9. Synthesis of pollution control flowsheet

If waste minimization via pollution prevention is not feasible, pollution control must be used. A combination of pollution prevention and pollution control is used in most of the cases, since there is almost always unavoidable waste generation dictated by the process. Pollution control consists of: Out-of-process recycling (on-site and off-site), waste treatment, and disposal. The objective of pollution control is to reduce the environmental load of the waste streams, prior to their discharge or disposal.

There is a variety of unit operations that can be used for pollution control. Therefore there are many alternative flowsheets that can be generated. A heuristic algorithm can be employed to reduce the number of alternatives.

10. Analysis of pollution control flowsheet

Once the pollution control flowsheet has been synthesized, i.e. the key unit operations have been laid down, its operating conditions and parameters have to be specified. Process simulation can be used afterwards to analyze the pollution control flowsheet.
11. Quantification of environmental loads

Based on the simulation results of the pollution control process, the environmental load of the discharge streams is quantified using appropriate entities (see step 3).

12. Is environmental load within acceptable limits?

Once the environmental load for each discharge stream in the pollution control flowsheet is quantified, the designer determines if the loads satisfy some pre-specified constraints.

If the loads are within the regulatory, operational, etc., constraints, the pollution control process is accepted as it successfully accomplishes waste minimization.

If not, a new pollution control process should be synthesized, analyzed and evaluated. The loop continues until a satisfactory answer to the pollution control problem is found or until financial, time or technology constraints are violated.

The pollution control part of the methodology described in this chapter, has been the main focus of this research. Computer-aided tools have been developed that assist in:

- synthesis of pollution control flowsheets,
- analysis of these flowsheets, and
- quantification of environmental loads.

The description of these tools is presented in Chapters 4-8.
CHAPTER 4

SYNTHESIS OF POLLUTION CONTROL FLOWSHEETS;
DEVELOPMENT OF WASTEMIN EXPERT

Although pollution prevention is the preferred waste minimization path, sometimes it may not be feasible or cost efficient. Subsequently, pollution control is the remaining option, and a pollution control flowsheet should be synthesized. Since a variety of pollution control unit operations are available, the synthesis problem is combinatorial, making the selection of a flowsheet a non-trivial task. The time and effort involved in the selection process can be reduced significantly by using domain-specific design knowledge.

Knowledge-based expert systems (KBES), which are programs derived from applied AI, provide the means to capture, represent and reproduce the required knowledge. As part of this dissertation, WasteMin Expert, an expert system prototype, has been developed to assist in the synthesis of pollution control flowsheets. WasteMin Expert takes as input information on a waste stream, and based on the design heuristics in its knowledge base, assists the user to interactively develop a flowsheet for pollution control.

General information on KBES, the synthesis algorithm of WasteMin Expert, and technical information on its development are presented in this chapter.

4.1 Knowledge-Based Expert Systems

KBES are programs that solve ill-defined problems, using domain-specific knowledge that is described by representation techniques and manipulated by strategic problem-solving approaches. There are three main components in an expert system (White and Goldsmith, 1990):
- The knowledge base, is the database that holds the knowledge needed to solve problems in a specific domain. That knowledge may be declarative (e.g., generic facts, rules of thumb, etc.) or procedural (e.g. relationships among the facts and how to use them). Therefore, the knowledge base of an expert system goes beyond the traditional concept of a database, for it contains not just static data, but also associations and dependencies among the data.

- The knowledge representation schemes, are the techniques used to represent and organize the knowledge contained in the knowledge base. The most widely used knowledge representation scheme is that of the production rules, i.e., conditional if-then descriptions of a given situation. Each rule has a condition or premise part (left hand side, LHS) and an action or conclusion part (right hand side, RHS). A rule “fires” when all the conditions in the premise part are satisfied by the current conditions. Then, the right hand side of the rule specifies the actions to be taken.

- The inference engine, defines a conceptual representation of the problem and a control strategy on how to solve it. The inference engine's objective is to know where to begin the problem-solving process, how to reason, traverse, or search its way through the problem by applying information in the knowledge base, and how to resolve any conflicts, inconsistencies, or other obstacles that might arise.

Expert systems can be built using either an AI computer language (e.g., LISP, Prolog) or an expert system shell. The development of an expert system using an AI language provides greater flexibility but requires more ground work. On the other hand, expert system shells usually provide an inference engine, and the user interface and knowledge representation facilities to readily develop customized knowledge bases. Most expert systems are developed using expert system shells.
4.2 Synthesis Algorithm of WasteMin Expert

WasteMin Expert carries out the waste minimization process synthesis task based on the concept of hierarchical decomposition, i.e., the main task is decomposed into smaller, easier to handle, sub-tasks. The high-level decision diagram of WasteMin Expert, shown in Figure 4.1, presents the main tasks of the system.

Figure 4.1 Abstract Decision Diagram of WasteMin Expert
The system starts with the specification of the input waste stream (total flowrate and composition). The required physical, chemical and biological properties of the compounds present in the waste stream are retrieved from the external component database. The user supplies any properties that are not available in the database.

Flowsheet synthesis starts with the potential solvent recovery step. WasteMin Expert recognizes opportunities for solvent recovery based on the number of solvents present in the waste stream, and the solids content of the stream. The decision diagram of the recovery task is shown in Figure 4.2. The rules used in this phase have been adapted from Venkataramani et al. (1992). When feasible, WasteMin Expert assumes a 90% recovery rate for each solvent.

Figure 4.2 Solvent Recovery Decision Diagram in WasteMin Expert
After recovery, the stream enters the pre-treatment phase (Figure 4.3). During pre-treatment, large suspended solid components are removed by screening, followed by an optional equalization step. The need for equalization is decided by the user.

![Figure 4.3 Pre-treatment Decision Diagram in WasteMin Expert](image)

The primary treatment section of the flowsheet is synthesized next. Neutralization and primary sedimentation (with or without coagulation, for enhanced performance) are the unit operations considered in this phase, as shown in Figure 4.4.
Figure 4.4 Primary Treatment Decision Diagram in WasteMin Expert
The secondary treatment phase consists exclusively of biological unit operations. The decision diagram of this phase is shown in Figure 4.5. The synthesis of the secondary treatment phase is based on the decision by the user to employ either suspended or attached growth biological processes. If the choice is made to use attached growth processes, the selection of the specific unit operation is based on the hydraulic loading rate, which is the ratio of the stream flowrate and the loading area. If the user selects suspended growth processes, the treatment unit is determined based on available space considerations.

Figure 4.5 Secondary Treatment Decision Diagram in WasteMin Expert
The tertiary (or advanced) treatment phase is optional. Tertiary treatment includes the removal of nitrogen, phosphorus, heavy metals, and wastewater disinfection. The decision diagram for this phase is shown in Figure 4.6.

The sludge treatment phase consists of thickening, stabilization, conditioning, and dewatering. The synthesis of the flowsheet is based on the decision diagrams shown in Figures 4.7a and 4.7b. The choice of unit operations for thickening and conditioning is based on whether the user wants to combine the sludges from the primary and secondary treatment steps. The sludge stabilization method is decided by the flowrate of the sludge stream. Decisions about dewatering are based on whether the sludge has been conditioned, as well as user preference on filtration type.

In the final synthesis step, WasteMin Expert considers alternatives for sludge disposal (Figure 4.8). The major decision point is whether the user wants to use combustion for sludge disposal. If the choice to use combustion is made, the specific disposal method is based on the water content of the sludge.

Finally, the unit operations selected during flowsheet synthesis are displayed on screen, saved in output files and the component database is updated, if needed.

Since WasteMin Expert is an interactive environment, any output of this system is the result of the algorithm and user input. A typical user of WasteMin Expert should be familiar with the basic principles of wastewater treatment design, in order to aid the system in several decision points.

The output of WasteMin Expert provides an ordered list of the main unit operations necessary to accomplish waste minimization. This high-level list is by no means complete; for example, auxiliary units such as pumps are not included. Also, more detailed design features, such as loops, stream splits, control systems, and processing conditions, are not addressed. The application of WasteMin Expert in the synthesis of a waste control flowsheet is demonstrated in Chapter 9.
Figure 4.6 Tertiary Treatment Decision Diagram in WasteMin Expert
Figure 4.7a Sludge Treatment Decision Diagram in WasteMin Expert
Figure 4.7b Sludge Treatment Decision Diagram in WasteMin Expert
4.3 Technical Description of WasteMin Expert

This section presents briefly, technical information about the actual computer implementation of the system and its knowledge base. Detailed documentation about the implementation and maintenance of WasteMin Expert as well as the complete listing of the rules, can be found in (Hood, 1994).

4.3.1 Computer Implementation

WasteMin Expert is an interactive knowledge-based system. The system asks the user a series of questions concerning the design characteristics of various unit operations. WasteMin Expert has been developed on a Macintosh personal computer using the
NEXPERT OBJECT expert system shell (Neuron Data Inc., 1991). The development of WasteMin Expert required customization of the shell in order to:

- ask the user questions in a “natural language” format,
- include numerical subroutines,
- read from and write to external files, and
- test for the presence of certain chemical elements (e.g., heavy metals).

4.3.2 Knowledge Base Structure

The knowledge stored in the knowledge base of WasteMin Expert is in the form of production rules. Each rule consists of:

- the rule name,
- a list of expressions for the premise (LHS), and
- a list of expressions for the conclusions (RHS).

The 106 rules contained in the knowledge base of WasteMin Expert can be categorized conceptually in three sets:

(i) **Control rules**: These rules are responsible for the operation and execution of the entire expert system. This top-level set controls the applicability of all other rules.

(ii) **Synthesis rules**: These rules carry out the synthesis of the waste minimization flowsheet. They are responsible for all design decisions. The knowledge acquisition for the synthesis rules was based on several texts and papers (Qasim, 1994; Venkataramani et al., 1992; Metcalf & Eddy Inc., 1991; Corbitt, 1990; Eckenfelder, 1989; Freeman, 1989; Wentz, 1989; Water Environment Federation/American Society of Civil Engineers, 1982; Sunstrom and Klei, 1979).

(iii) **Auxiliary rules**: These rules are responsible for the execution of numerical subroutines and the generation of output report files.
The ordered list of unit operations generated by WasteMin Expert, can serve as a good initial design for the synthesis of a pollution control facility using other computer-aided tools, such as a process simulator. In the simulation environment, the user can make decisions on more detailed issues such as process conditions, loops, and process economics. The development of such a simulator is presented in the following chapter.
CHAPTER 5

ANALYSIS OF POLLUTION CONTROL PROCESSES;
DEVELOPMENT OF ENVIROCAD

Following process synthesis, analysis is the second major activity of process design. The scope of process analysis is the close examination and evaluation of the alternative designs generated through synthesis. Analysis is carried out by means of process simulation, mainly in the form of mass and energy balancing, sizing and costing calculations. Computer simulation has been used very successfully in traditional chemical engineering applications for more than thirty years. The major benefits of simulation include: the identification of key process parameters, the investigation of their effect on the overall process, and the rapid evaluation of different process alternatives. Process simulation is currently emerging as a valuable assisting tool for waste treatment and disposal processes.

In addition to the features that any process simulator should have (e.g., graphical flowsheet representation and economic evaluation capabilities), a simulator used for waste treatment and disposal processes should be able to:

- Model the unique unit operations used in pollution control (e.g., equalizers, clarifiers, aeration basins, trickling filters, digestors, dryers, adsorbers, strippers, cyclones, fabric filters and incinerators), predict their performance, and estimate their capital and operating costs.
- Support the quantification of environmental loads and the representation of the lumped properties (e.g., COD) associated with the waste streams.
- Handle new types of materials, such as the different forms of solids resulting from the biological treatment processes, that are not well characterized.
• Provide features for environmental impact assessment, such as Life Cycle Analysis and Total Cost Assessment.

The development of a simulator, featuring many of the above characteristics, was a major part of the present dissertation. This chapter presents the technical description, user interface, and economic evaluation capabilities of EnviroCAD, the simulator developed for the analysis and evaluation of pollution control processes.

5.1 Technical Description of EnviroCAD

The development of EnviroCAD was based on the infrastructure of the analytic component of BioDesigner (Petridis, 1990), a computer-aided process design tool for integrated biochemical processes. EnviroCAD is a steady-state process simulator. It has been developed on the Apple Macintosh computer platform, and it follows the sequential modular approach for simulation. EnviroCAD is written using the THINK C™ compiler (Symantec Inc., 1993) which is fully compatible with the ANSI C standards. The C programming language supports the representation and manipulation of advanced data structures. A data structure is a "compound" variable type that contains an arbitrary group of related data (members).

Data structures are well suited for the needs of process simulation, since they can easily represent key flowsheet components, such as unit operations, chemical components and process streams. The main data structures of EnviroCAD, are:

• the two unit operation structures,
• the component structure, and
• the stream structure.

Various instances of these structures are combined to form a flowsheet. Memory for these instances is allocated dynamically during program execution, allowing a flowsheet
to have any number of components, unit operations and streams (the memory capacity of the computer used, is the only limitation).

The dynamic memory allocation, as well as the manipulation and linkage of the data structures in order to form a flowsheet, are accomplished by the use of *pointers*, which are among the most sophisticated features of the C language.

### 5.1.1 The Unit Operation Structures

There are two data structures associated with each unit operation in EnviroCAD (see Figure 5.1). The top data structure (*UnitOper*), contains elements common to all unit operations. For example, each unit operation has a name, a picture, a power requirement, etc. The structure in the lower level, contains information specific to each unit operation. For instance, the *Clarifier* structure, shown in Figure 5.1, contains elements unique to the clarifier model, such as its volume and clarification area. The *UnitOper* structure accesses the information of the *Clarifier* structure, through the *theModel* pointer. Now, the *Clarifier* structure, in addition to its own members, includes the members of the *UnitOper* structure. A complete listing of the unit operation data structures contained in EnviroCAD can be found in Appendix I.

Figure 5.2 shows the hierarchy of the unit operation structures in EnviroCAD. *UnitOper* is the top structure, followed by a lower level of unit operation structures (*equalizer, clarifier, aeration basin*, etc.). The structures in the next level (triangles in Figure 5.2), correspond to specific instances of each unit operation in a flowsheet. Note that there is only one top level structure (*UnitOper*), but there are as many data structures in parallel in the lower level, as there are unit operation models in EnviroCAD. A unit operation model in EnviroCAD refers to a collection of equations and subroutines used to model the unit operation and, in effect, defines the calculation
of outlet stream variables from inlet stream information and engineering specifications.

The unit operation models are presented analytically in Chapter 7.

The unit operation structures of a flowsheet form a linked list. A pointer is placed to the head of this list, that can be used to traverse the list and access the members of any unit operation structure. This linked list is shown schematically in Figure 5.5.

---

**Figure 5.1** The Unit Operation Data Structures
5.1.2 The Component Structure

The Component data structure, shown in Figure 5.3, stores information about the properties of each component present in a flowsheet. In addition to traditional component properties (e.g. molecular weight, density, etc.), component specific properties that facilitate the calculation of lumped stream properties (e.g., TOC, COD, ThOD, etc.) are also included in this data structure. The various instances of the component structure in a flowsheet, are stored in the form of a list (see Figure 5.5).

```
Structure Component

char Name[32];
float MW;
float Density;
float HeatCapacity[4];
float Tb;
float Tc;
float Pc;
float Zc;
float Omega;
float Antoine[3];
float HeatOFFormation;
float Henry;
float LogPow;
.
.
.
float TOC;
float COD;
float ThOD;
float BOD_COD_Ratio;
float BOD_Ratio;
float TKN;
float NH3_Fraction;
float NO3_NO2;
float TP;
float TS;
float TSS_TS_Ratio;
float VSS_TSS_Ratio;
float DVSS_VSS_Ratio;
float VDS_TDS_Ratio;
float DVDS_VDS_Ratio;
```

Figure 5.3 The Component Data Structure
5.1.3 The Stream Structure

The Stream data structure is shown in Figure 5.4. Some members of this structure are used for the storage of the lumped stream properties (e.g., TOC_Stream, COD_Stream, etc.), while others facilitate the life cycle inventory analysis (e.g., m_LCAPproduct, m_LCACoal, etc.) in EnviroCAD. The many instances of the Stream structure in a flowsheet, form a linked list (see Figure 5.5).

![Structure Stream](image)

**Figure 5.4 The Stream Data Structure**
Streams represent the flow of material from one unit operation to the next. The St_Comp structure is used to store the mass flowrate of each component in a stream. Each Stream structure has several St_Comp structures linked to it. The number of the St_Comp structures is equal to the number of the Component structures of the flowsheet.

5.1.4 Flowsheet Representation in EnviroCAD

A flowsheet in EnviroCAD consists of a collection of instances of the UnitOper, Stream and Component structures. The instances of the same structure type form an individual linked list. Figure 5.5 presents schematically the structure of an EnviroCAD flowsheet. There are pointers to the heads of the unit operations linked list, the streams linked list, and the components list, for accessing the information contained in each one of the instances of the data structures.

The connectivity of a flowsheet is established by using information stored in the UnitOper and Stream data structures (shown in Figures 5.1 and 5.4 respectively).

The Stream structure stores the names of the source (from_unit[16]) and destination (to_unit[16]) unit operations. A stream without source unit operation is termed a feed stream, while a stream without destination unit operation is a product stream. Finally, a stream which has both a source and a destination unit is an intermediate stream.

Similarly, in the UnitOper structure (Figure 5.1), there are members (e.g., FeedHPoints, ProdHPoints) that are used to store information about all the streams that are associated with each unit operation.
Figure 5.5 Flowsheet Structure in EnviroCAD
5.2 User Interface of EnviroCAD

EnviroCAD uses a graphical interface to enhance the human/computer communication and reduce the learning period. A process flowsheet is represented graphically on the screen of the simulator. All input/output information is provided/displayed through dialog windows. The user interface is presented in more detail in Chapter 8. EnviroCAD also features on-line help.

5.3 Economic Evaluation in EnviroCAD

For a pollution control process, EnviroCAD calculates the purchase cost of equipment, the fixed capital investment, and the annual operating cost. Equipment cost is estimated as a function of equipment capacity, materials of construction, and certain design characteristics. The correlations used in EnviroCAD for purchase cost estimation, were derived from curve-fitting of price data that appear in the literature (Cooper and Alley, 1994; Peters and Timmerhaus, 1991; Garrett, 1989; U. S. EPA, 1980).

The fixed capital investment is calculated based on the total purchase cost of equipment using multipliers (Valle-Riestra, 1983). The annual operating cost includes cost of consumables, labor, utilities, equipment depreciation, maintenance, and disposal of ultimate waste. The economic evaluation capabilities of EnviroCAD are demonstrated in Chapter 9.

The quantification of environmental loads, which is an important feature of EnviroCAD, is presented in detail in the following chapter.
CHAPTER 6

QUANTIFICATION OF ENVIRONMENTAL LOADS IN ENVIROCAD

Process design with environmental considerations is facilitated by the quantification of pollutant release to the environment (environmental loads). Waste minimization alternatives can be evaluated based on the calculated environmental loads for a process. In order to provide a common ground for comparison of various processes, quantifiable entities which can be estimated for every process need to be used. These entities should represent waste generation in a process, and provide a measure of success of waste minimization efforts. EnviroCAD supports the calculation of:

- Pollution inventories.
- Eco-vectors.
- Lumped environmental stream properties.

Pollution inventories and eco-vectors are mainly associated with manufacturing processes. Hence, they are very useful during the pollution prevention part of the design methodology presented in Chapter 3. On the other hand, the environmental stream properties are used for the characterization of manufacturing as well as pollution control processes. The features for environmental load quantification in EnviroCAD are presented in the following sections.

6.1 Pollution Inventories

Figure 6.1 depicts the flow of mass in a manufacturing process. Raw materials ($\Sigma R_i$) are converted into products ($\Sigma P_j$), while wastes ($\Sigma W_k$) and emissions ($\Sigma E_m$) are generated. The calculation of the total amounts of pollutants generated from a process ($\Sigma W_k + \Sigma E_m$) is an absolute measure of pollution. The waste generated ($\Sigma W_k$) can be
calculated based on the overall material balance of the process, while for the estimation of emissions ($\sum E_m$) there are available methods and correlations (see Section 6.1.1). Such measures are straightforward and easily quantifiable. Their major disadvantage is that they are directly related to the production level (i.e., a decrease in the waste generation might simply result from a reduction in production and not from a successful waste minimization effort). The solution to this problem is to index waste generation to some measure of production.

![Figure 6.1 Representation of Mass Flow in a Manufacturing Process](image)

For example, an index can be defined as the total amount of pollutants generated per amount of product produced:

$$\frac{\sum W_k + \sum E_m}{P_j}$$

Depending on the relevant application, other indices may be defined based on different process inputs and outputs.

Waste minimization can be greatly facilitated by using the information supplied from the overall mass balance of a process. The waste minimization efforts should focus mainly on the unit operations with the higher generation of pollutants. The amount of
each pollutant that is present in a process output stream is reported in the stream report of EnviroCAD (see Chapter 8).

6.1.1 Calculation of Process Emissions
Emissions pose a major threat to the environment because they are released directly, in contrast with liquid and solid wastes which are generally treated before disposal. Emissions can be classified into four major categories:

- **Process Emissions**, resulting from processing equipment and representing the major component of total emissions. EnviroCAD is equipped with a module for calculating process emissions.

- **Storage and Handling Emissions**, associated with storage and transfer of solvents.

- **Fugitive Emissions**, unintentional releases from pumps, valves, flanges and other connectors. Since it is very difficult to measure directly fugitive emissions, estimation methods have been developed for this purpose (Schaich, 1991).


Process emissions in EnviroCAD are calculated based on the guidelines set by EPA (1978). The basic equations and assumptions governing these calculations are:

(i) Ideal vapor and liquid phases,

\[ P_i = y_i P = x_i P_i^s \]

(ii) Ideal gas law,

\[ P V = n R T \]
(iii) Antoine equation for the calculation of vapor pressures,

\[ \log_{10} P_i^s = a - \left( \frac{b}{c + T} \right) \]

(iv) Dalton's law,

\[ P = \sum P_i \]

where \( y_i \) = fraction of component \( i \) in the gas phase,

\( x_i \) = fraction of component \( i \) in the liquid phase,

\( P_i^s \) = vapor pressure of component \( i \),

\( P_i \) = partial pressure of component \( i \),

\( P \) = total pressure of the system,

\( V \) = system volume,

\( n \) = total moles of chemicals in the system,

\( R \) = universal gas constant,

\( T \) = absolute system temperature, and

\( a, b, c \) = Antoine constants.

Process emissions result from the following operations: charging, evacuation, nitrogen or air sweep, heating, gas evolution, vacuum operation, and drying (U. S. EPA, 1978). These operations are part of any process unit in a manufacturing process. The sum of all emissions calculated for each operation in each unit step is presented in an emissions report in EnviroCAD. The source code for the calculation of process emissions is contained in Appendix III. For illustration purposes, the algorithm for calculating emissions resulting from charging chemicals into a vessel is presented.
When a liquid is charged into a vessel, vapors are discharged by simple volume displacement. These vapors stem from the incoming stream and the vessel contents, if any. It is assumed that: (i) the volume of gas displaced from the vessel is equal to the volume of liquid charged into the vessel, and (ii) the air displaced from the vessel is saturated with the chemical vapor(s) at the exit temperature. Subject to these assumptions, the process emissions are calculated using the following algorithm:

- Specification (or calculation based on charge rate) of the rate of air displacement $V_r$ [m$^3$/h], and of the exit temperature $T$ [K].
- Calculation of the mole fraction of each chemical in the liquid phase, $x_i = \frac{n_i}{n}$.
- Calculation of the vapor pressure $P_{i}^s$ [mm Hg] of each chemical.
- Calculation of the amount of each chemical emitted $E_i$ [kg/h], using the following modified form of the ideal gas law:

$$E_i = \frac{x_i P_{i}^s V_r MW_i}{R T}$$

where $MW_i =$ molecular weight of component $i$ [kg/kgmole].

6.2 Life Cycle Inventory Analysis

Life Cycle Analysis (LCA) is a method of estimating the environmental load of a product, process or activity during its entire life cycle, analyzing impact on the environment, and recommending improvements. LCA consists of four steps (Society of Environmental Toxicology and Chemistry, 1993):

- **Goal Definition and Scoping.** Definition of the aim, the boundaries of the system, the target audience, the data requirements and their degree of confidence.
- **Inventory Analysis.** Estimation of the environmental loads associated with the entire life cycle of a product or process.
- **Impact Assessment.** Characterization and evaluation of the effects to human health and to the environment of the loads identified in the inventory step.

- **Improvement Assessment.** List of process or product modifications aimed at reducing the environmental releases and/or their effects.

These four steps are interconnected, as shown in Figure 6.2. The inventory and the impact assessment steps can generate information which can form the basis for process or product redesign.

![Technical Framework for Life Cycle Analysis](image)

**Figure 6.2** Technical Framework for Life Cycle Analysis

The identification and quantification of all the environmental loads in the complete life cycle of a product, is a labor intensive task, due to the requirement of data as early as raw materials extraction. However, it is always possible to evaluate the loads generated by a manufacturing process itself; these loads can then be added to those of the input streams and result in the environmental loads of the process outputs.
The objective of the inventory analysis is to estimate the total environmental loads associated with the life-cycle of a product or process. Based on the estimated loads, an environmental impact assessment can be made, and improvements aiming at reducing the environmental impact can be suggested. EnviroCAD facilitates the inventory analysis step of LCA, by using the concept of the eco-vector (Castells et al., 1994). An eco-vector $v$ is a multidimensional vector, each dimension of which corresponds to a particular environmental load or depleted natural resource. There are no standards available regarding the environmental loads that should be contained in an eco-vector. An example of an eco-vector is presented in the following equation.

$$
v = \begin{bmatrix}
\text{Fuels} \\
\text{Air emissions} \\
\text{Liquid discharges} \\
\text{Solid waste}
\end{bmatrix}
$$

Each process stream has an associated mass eco-vector $v_m$, whose elements are expressed in environmental load (EL) per unit mass of stream. There is no strict definition of EL units; an LCA study can be based on any units, as long as they can be balanced and accumulated. The product of the mass flow $M$ [mass/unit time] of the process stream times its corresponding vector $v_m$ gives the amount of pollutants $P_m$ [EL/unit time] generated up to this process step:

$$P_m = M v_m$$

Similarly, the energy flows of a process have an associated energy eco-vector $v_e$ [EL/unit energy]. The product of the energy flow $E$ [energy/unit time] and the corresponding vector $v_e$ yields the flow of pollutants $P_e$ [EL/unit time] associated with the energy:
$P_e = E v_e$

Note that $P_m$ and $P_e$ can be summed because they are both measured in units of EL per unit time.

![Figure 6.3 Generic Process Representation](image)

Based on the above notation, the overall EL balance for the generic process shown in Figure 6.3, is:

$$
\sum_i (R_i v_{m,i}) + \sum_n (E_n v_{e,n}) = \sum_k (W_k v_{m,k}) + \sum_j (P_j v_{m,j})
$$

where

- $R_i$ = mass flow of raw material stream $i$,
- $P_j$ = mass flow of product stream $j$,
- $W_k$ = mass flow of waste stream $k$,
- $E_n$ = energy input $n$,
- $v_{m,i}$ = mass eco-vector of raw material stream $i$,
- $v_{m,j}$ = mass eco-vector of product stream $j$. 

\( v_{m,k} = \) mass eco-vector of waste stream \( k \), and

\( v_{e,n} = \) energy eco-vector of input \( n \).

The mass and energy eco-vectors have been implemented in EnviroCAD, as shown in Figures 6.4 and 6.5 respectively. The list of environmental load components (fuel inputs, emissions, liquid discharges, solid waste) contained in the current implementation of the eco-vectors is not meant to be exhaustive by any means; it can be modified as needed.

![Figure 6.4 Implementation of Mass Eco-vector in EnviroCAD](image-url)
For the application of the life cycle inventory analysis in EnviroCAD a process flowsheet is required. Each process stream is classified as feed, product or waste (intermediate streams are specified as product streams). For each feed and waste stream, the user provides the environmental load data in terms of mass entries in the respective eco-vector slots. For waste streams, the entries in the eco-vector must be negative (this convention is required to close the EL balance). The energy eco-vectors should also be specified. While there is one mass eco-vector for each material stream, there are only two energy eco-vectors (one associated with electricity and one with steam) currently.
Upon specification of all input eco-vectors, the simulator calculates the environmental loads of the product stream(s) and generates a report with the results. In the current implementation, if there are more than one product streams, the distribution of the loads among them is based on their relative mass flowrates. The source code for the life cycle inventory module in EnviroCAD is presented in Appendix IV.

6.3 Environmental Stream Properties

Based on the composition of each stream, EnviroCAD calculates and displays a number of environmental stream properties (Figure 6.6). These properties are used routinely to characterize liquid wastestreams, and regulate the discharges to large liquid bodies (e.g., rivers and lakes). The list of environmental stream properties contained in EnviroCAD is shown in Table 6.1.

<table>
<thead>
<tr>
<th>Environmental Properties</th>
<th>mg/L</th>
<th>kg/day</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOC:</td>
<td>2208.388</td>
<td>13499.489</td>
</tr>
<tr>
<td>COD:</td>
<td>7496.072</td>
<td>45822.175</td>
</tr>
<tr>
<td>ThOD:</td>
<td>7785.191</td>
<td>47589.509</td>
</tr>
<tr>
<td>BODu:</td>
<td>5911.131</td>
<td>36133.711</td>
</tr>
<tr>
<td>BODs:</td>
<td>3840.008</td>
<td>23473.299</td>
</tr>
<tr>
<td>TKN:</td>
<td>251.813</td>
<td>1539.288</td>
</tr>
<tr>
<td>NH3: 67.1 % TKN Fraction</td>
<td>169.032</td>
<td>1033.265</td>
</tr>
<tr>
<td>Nitrates-Nitrites:</td>
<td>12.349</td>
<td>75.486</td>
</tr>
<tr>
<td>Total Phosphorus (TP):</td>
<td>12.826</td>
<td>78.400</td>
</tr>
<tr>
<td>TSS:</td>
<td>2400.913</td>
<td>14676.360</td>
</tr>
<tr>
<td>VSS: 89.6 % TSS Fraction</td>
<td>2150.274</td>
<td>13144.247</td>
</tr>
<tr>
<td>DVSS: 90.0 % VSS Fraction</td>
<td>1935.246</td>
<td>11829.822</td>
</tr>
<tr>
<td>TDS:</td>
<td>516.003</td>
<td>3154.234</td>
</tr>
<tr>
<td>UDS: 0.0 % TDS Fraction</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>DVDS: 0.0 % UDS Fraction</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Figure 6.6 Output Dialog Window for Environmental Stream Properties
Table 6.1 Environmental Stream Properties in EnviroCAD

<table>
<thead>
<tr>
<th>Stream Property Name</th>
<th>EnviroCAD Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Organic Carbon</td>
<td>TOC</td>
</tr>
<tr>
<td>Chemical Oxygen Demand</td>
<td>COD</td>
</tr>
<tr>
<td>Theoretical Oxygen Demand</td>
<td>ThOD</td>
</tr>
<tr>
<td>Biochemical Oxygen Demand (5-day)</td>
<td>BOD_5</td>
</tr>
<tr>
<td>Biochemical Oxygen Demand (ultimate)</td>
<td>BOD_u</td>
</tr>
<tr>
<td>Total Kjeldahl Nitrogen</td>
<td>TKN</td>
</tr>
<tr>
<td>Ammonia Nitrogen</td>
<td>NH_3</td>
</tr>
<tr>
<td>Nitrite / Nitrate Nitrogen</td>
<td>NO_2/NO_3</td>
</tr>
<tr>
<td>Total Phosphorus</td>
<td>TP</td>
</tr>
<tr>
<td>Total Solids</td>
<td>TS</td>
</tr>
<tr>
<td>Total Suspended Solids</td>
<td>TSS</td>
</tr>
<tr>
<td>Total Dissolved Solids</td>
<td>TDS</td>
</tr>
<tr>
<td>Volatile Suspended Solids</td>
<td>VSS</td>
</tr>
<tr>
<td>Fixed Suspended Solids</td>
<td>FSS</td>
</tr>
<tr>
<td>Degradable Volatile Suspended Solids</td>
<td>DVSS</td>
</tr>
<tr>
<td>Non Degradable Volatile Suspended Solids</td>
<td>NDVSS</td>
</tr>
<tr>
<td>Volatile Dissolved Solids</td>
<td>VDS</td>
</tr>
<tr>
<td>Fixed Dissolved Solids</td>
<td>FDS</td>
</tr>
<tr>
<td>Degradable Volatile Dissolved Solids</td>
<td>DVDS</td>
</tr>
<tr>
<td>Non Degradable Volatile Dissolved Solids</td>
<td>NDVDS</td>
</tr>
</tbody>
</table>

The relationship among the various forms of nitrogen and solids in EnviroCAD is shown in Figures 6.7 and 6.8 respectively.
The calculation of all environmental stream properties is based explicitly on the mass contribution of individual chemical components in these properties (see Appendix V). The contribution factors for each component are supplied to the program through a series of appropriate dialog windows such as the one shown in Figure 6.9, during the initialization of the flowsheet (see Chapter 8).
Some of these contribution factors (e.g., TOC, ThOD, TP) are derived from theoretical stoichiometric calculations, while others (e.g., BOD₅/BOD₅ ratio) are based on experimental data found in the literature (Verschueren, 1983). For example, the TOC contribution of any compound is calculated theoretically as the ratio of the total weight of carbon in the molecule divided by the molecular weight of the compound. In the case of methanol (CH₃OH, MW = 32.04), this calculation yields:

$$\text{TOC}_{\text{methanol}} = \frac{12.01}{32.04} = 0.37 \frac{\text{g C}}{\text{g methanol}}$$

Similarly, the ThOD contribution of each compound is calculated based on the amount of oxygen needed for the complete chemical oxidation of the compound. For the complete oxidation of the hydrocarbon CₙHₘ to carbon dioxide and water,
\[ C_nH_m + \left(n + \frac{m}{4}\right)O_2 \rightarrow nCO_2 + \frac{m}{2}H_2O \]

the ThOD of \(C_nH_m\) is:

\[
\text{ThOD} = \frac{32 \left(n + \frac{m}{4}\right)}{12.01 \left(n + \frac{m}{4}\right)}
\]

When the organic molecule contains other elements, such as N, S, P, etc., the ThOD depends on the final oxidation stage of these elements.

EnviroCAD carries out material balances on individual components and estimates the lumped stream properties based on stream composition. If the contribution of component \(i\) to the environmental property \(p\) is \(c_{ip}\), the lumped stream property \(c_p\) is:

\[
c_p = \sum_{i} (m_i c_{ip})
\]

where \(m_i\) is the mass flowrate of component \(i\) in the stream.

Environmental stream properties can be used as a criterion for ranking waste stream in terms of pollution severity. Based on the above equation, high values of environmental properties may result from either high component flowrate and/or high contribution from an individual component. The calculation of stream properties based on contribution of individual components is an important feature because it allows the tracking of the fate of hazardous chemicals in integrated pollution control processes. The availability of this information forms the basis of process retrofitting, while placing emphasis on the source(s) of the worse pollutants.
UNIT OPERATIONS IN ENVIROCAD

Unit operations constitute the core component of EnviroCAD because they carry out a major part of the simulation. A unit operation is represented on the computer screen with an icon. Associated with each unit operation are one or more input streams and one or more output streams. A mathematical model describes the performance of each unit operation. Based on the input stream variables and the design specifications, unit operation models size the unit, calculate the output stream variables, and estimate the purchase cost of the unit. The source code for these models is presented in Appendix II.

For presentation purposes, the unit operations are categorized based on their main function:

- Physical wastewater treatment,
- Chemical wastewater treatment,
- Biological wastewater treatment,
- Sludge treatment, and
- Air pollution control.

The description of each unit operation starts with its icon, input and output process streams. The input and output data of the process model follow. Finally, under the generic title Model, the assumptions, equations and solution algorithm, are described.

7.1 Physical Wastewater Treatment Unit Operations

Physical treatment unit operations are used for flow and concentration equalization, and removal of suspended solids and volatile organic compounds (VOCs) from wastewater. The removal of these pollutants takes place through the application of physical forces.
7.1.1 Equalization

Equalization is used to overcome the problems caused by variations in flowrate and BOD concentration, to improve the performance of the downstream processes, and to reduce the size and cost of downstream treatment facilities (Metcalf & Eddy Inc., 1991).

**Input Data:**
- Number of time periods, \( N \) [dimensionless].
- Duration of time period \( i \), \( t_i \) [min].
- Influent volumetric flowrate during time period \( i \), \( Q_i \) [m\(^3\)/min].
- Influent concentration during time period \( i \), \( C_i \) [mg/L].
- Equalization volume \( V \) [m\(^3\)]:
  (i) Specified directly by the user, or
  (ii) Estimated by the system.

**Output Data:**
- Equalized volumetric flowrate, \( Q \) [m\(^3\)/min].
- Equalized effluent concentration during time period \( i \), \( C_{E_i} \) [mg/L].
- Influent concentration peaking factor, \( PF_{in} \) [dimensionless].
- Equalized concentration peaking factor, \( PF_{ef} \) [dimensionless].

If equalization volume is estimated by the system:
- Equalization volume, \( V \) [m\(^3\)].
Model:

The assumption made for the equalization model is that the influent flowrate and concentration during each time period remain constant. The total equalization time is:

\[ t = \sum_{i=1}^{N} t_i \]

The total influent cumulative volume is:

\[ V_{\text{cum}} = \sum_{i=1}^{N} (Q_i t_i) \]

The equalized volumetric flowrate is:

\[ Q = \frac{\sum_{i=1}^{N} Q_i}{N} \]

The average influent concentration is estimated by:

\[ C_{\text{avg}} = \frac{\sum_{i=1}^{N} (C_i Q_i t_i)}{V_{\text{cum}}} \]

Note that since no transformations are taking place during equalization, the average influent concentration is also the average effluent concentration.

The influent peaking factor is:

\[ PF_{\text{in}} = \frac{C_{\text{max}}}{C_{\text{avg}}} \]
where $C_{\text{max}} = \max(C_i)$.

If the equalization volume $V$ is specified directly by the user, the effluent concentration from the equalizer at the end of period $i$, is calculated based on the following material balance (Eckenfelder, 1989):

$$CE_i = \frac{C_i Q_i t_i + CE_{i-1} V}{Q_i t_i + V}$$

In the above equation, the assumption for the first time period ($i=1$) is that $CE_0 = C_{\text{avg}}$. If the system estimates the equalization volume, the following procedure is used:

For each time period $n$ ($1 \leq n \leq N$), the following two variables are calculated

$$V_n^+ = \left[ \sum_{i=1}^{n} (Q_i t_i) - Q_n \right], \quad \text{if} \quad \sum_{i=1}^{n} (Q_i t_i) \geq Q_n$$

$$V_n^- = \left[ Q_n - \sum_{i=1}^{n} (Q_i t_i) \right], \quad \text{if} \quad Q_n > \sum_{i=1}^{n} (Q_i t_i)$$

The equalization volume $V$ is then given by:

$$V = \max(V_n^+) + \max(|V_n^-|)$$

In this case, the effluent concentration at the end of period $i$, is given by:

$$CE_i = \frac{C_i Q_i t_i + CE_{i-1} V_i}{Q_i t_i + V_i}$$

Again, the assumption for the first time period ($i=1$) is that $CE_0 = C_{\text{avg}}$, and $V_i$ is the volume of wastewater at the tank at the end of each period, calculated by:
\[ V_i = V_{i-1} + (Q_i - Q) t_i \]

The effluent peaking factor is:

\[ PF_e = \frac{C_{E_{\text{max}}}}{C_{\text{avg}}} \]

where \( C_{E_{\text{max}}} = \max(CE_i) \).

The solution algorithm for the equalization model is shown in Figure 7.1.

```
Calculate total equalization time
Calculate total influent cumulative volume
Calculate equalized effluent volumetric flowrate
Calculate average influent concentration
Calculate influent peaking factor
If equalization volume is estimated by the system
   Calculate equalization volume
Calculate effluent concentration at each period
Calculate effluent peaking factor
Estimate cost
```

**Figure 7.1** Solution Algorithm for the Equalization Model

### 7.1.2 Discrete Particle Sedimentation

Sedimentation is the separation of suspended particles from wastewater, by gravitational settling. Discrete particle sedimentation refers to the phenomenon where solid particles settle individually without any significant interaction with neighboring particles.
Input Data:

- Mass flowrate of component i in the feed stream, \( F_i \) [kg/h].
- Sedimentation fraction of component i, \( r_i \) [dimensionless].
- Concentration of solid particles in sludge, \( C_{\text{solid}} \) [mg/L].
- Terminal settling velocity of the critical particle, \( V_c \) [m/h]:
  (i) Specified directly by the user, or
  (ii) Estimated by the system.
- Specification of:
  (i) Clarifier depth \( H \) [m], or
  (ii) Detention time \( t \) [h].
- Specification of clarifier geometry:
  (i) Circular, or
  (ii) Rectangular.

If the terminal settling velocity of the critical particle is estimated by the system:

- Particle diameter, \( d_p \) [\( \mu \text{m} \)].
- Particle density, \( \rho_p \) [g/cm\(^3\)].
- Liquid density, \( \rho \) [g/cm\(^3\)].
- Liquid viscosity, \( \mu \) [cp].

If the rectangular clarifier geometry has been selected:

- Ratio of clarifier length to clarifier width, \( R \) [dimensionless].

Output Data:

- Mass flowrate of component i in the clarified stream, \( C_i \) [kg/h].
- Mass flowrate of component i in the sludge stream, \( S_i \) [kg/h].
- Surface area of the clarifier, \( A \) [m\(^2\)].
- Volume of the clarifier, \( V \) [m\(^3\)].
If the terminal settling velocity of the critical particle is estimated by the system:

- Terminal settling velocity of the critical particle, $V_c$ [m/h].

If circular clarifier geometry has been specified:

- Clarifier diameter, $D$ [m].

If rectangular clarifier geometry has been specified:

- Clarifier length, $L$ [m].
- Clarifier width, $W$ [m].

*Model:*

In the design of discrete particle sedimentation, a critical particle with terminal settling velocity $V_c$ is selected. The basic design assumption is that all particles that have a terminal velocity equal to or greater than $V_c$ will be removed completely.

If the terminal settling velocity of the critical particle, must be estimated by the system, the calculation is based on the following procedure (McCabe et al., 1993).

At first, the settling constant $K$ is calculated:

\[
K = d_p \left[ \frac{g \rho (\rho_p - \rho)}{\mu^2} \right]^{1/3}
\]

where $g = 9.81 \text{ m/s}^2$.

The terminal settling velocity $V_c$ for the critical particle, is calculated from:

\[
V_c = \left[ \frac{4g d_p^{1+n} (\rho_p - \rho)}{3 b_1 \mu^n \rho^{1-n}} \right]^{1/(2-n)}
\]

The constants $b_1$ and $n$ are selected based on the constant $K$, which specifies the settling regime of the particle (see Table 7.1).
Table 7.1 Discrete Particle Settling Regimes

<table>
<thead>
<tr>
<th>$K$ Range</th>
<th>Regime</th>
<th>$b_1$</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K \leq 3.3$</td>
<td>Stokes</td>
<td>24</td>
<td>1.0</td>
</tr>
<tr>
<td>$3.3 &lt; K \leq 43.6$</td>
<td>Intermediate</td>
<td>18.5</td>
<td>0.6</td>
</tr>
<tr>
<td>$43.6 &lt; K \leq 2360$</td>
<td>Newton</td>
<td>0.44</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Once the settling velocity $V_c$ is known, the sedimentation area $A$ is calculated by:

$$A = \frac{Q}{V_c}$$

where $Q$ = volumetric flowrate of feed stream [m$^3$/h].

The clarifier depth and the detention time of the wastewater in the clarifier should be such that all particles with the critical velocity $V_c$ will settle to the bottom of the tank. The critical settling velocity, detention time, and clarifier depth are related as follows:

$$V_c = \frac{H}{t}$$

The sedimentation area is related to the geometrical characteristics of the clarifier by:

$$A = \pi \frac{D^2}{4} = LW = R W^2 = \frac{L^2}{R}$$

while the clarifier volume is: $V = A H$.

The mass flowrate of each component in the sludge stream is given by:

$$S_i = F_i r_i$$
while the mass flowrate in the clarified stream is:

\[ C_i = F_i (1 - r_i) \]

The solution algorithm for the Sedimentation model is given in Figure 7.2.

```plaintext
If critical settling velocity is specified by the user
    Calculate clarification area
else
    Calculate parameter K
    Calculate constants b1 and n
    Calculate critical settling velocity
    Calculate clarification area
If circular clarifier geometry is specified
    Calculate clarifier diameter
else
    Calculate clarifier length
    Calculate clarifier width
If clarifier depth is specified
    Calculate detention time
else
    Calculate clarifier depth
Calculate clarifier volume
Do material balances
Adjust liquid content in sludge stream
Estimate cost
```

**Figure 7.2** Solution Algorithm for the Discrete Particle Sedimentation Model

### 7.1.3 Dissolved-Air Flotation
Dissolved-air flotation is used to separate solid or liquid particles from wastewater. Separation is accomplished by dissolving pressurized air bubbles into the liquid phase. The bubbles attach to the particulate matter, and the buoyant force of the combined particle and air bubbles, causes the particles to rise to the surface. Skimming can be used afterwards, to remove the particles (Metcalf & Eddy Inc., 1991).

**Input Data:**
- Mass flowrate of component i in the feed stream, \( F_i \) [kg/h].
- Removal fraction of component i, \( r_i \) [dimensionless].
- Solubility of air in the wastewater, \( S_{air} \) [mL/L].
- Fraction of air dissolved at pressure \( P \), \( f \) [dimensionless].
- Surface loading rate, \( U \) [m\(^3\)/m\(^2\)-h].
- Concentration of solid particles in sludge, \( C_{solid} \) [mg/L].
- Specification of flotation type:
  (i) With recycle, or
  (ii) Without recycle.

If flotation without recycle:
- Specification of:
  (i) Air/Solid ratio, \( (A/S) \) [mL/mg], or
  (ii) Operating pressure, \( P \) [atm].

If flotation with recycle:
- Recycle ratio \( R \) [dimensionless]:
  (i) Specified by the user, or
  (ii) Estimated by the system.

If system estimates the recycle ratio:
- Air/Solid ratio.
- Operating pressure.

**Output Data:**
- Mass flowrate of component i in the clarified stream, \( C_i \) [kg/h].
- Mass flowrate of component i in the sludge stream, \( S_i \) [kg/h].
- Air requirement, \( A \) [m³/h].
- Surface area, \( D \) [m²].

In any other case, except when the system estimates the recycle ratio \( R \):
- Air/Solid ratio (A/S) [mL/mg], if operating pressure is specified.
- Operating pressure \( P \) [atm], if Air/Solid ratio is specified.

**Model:**

The surface area of the flotation unit is given by:

\[
D = \frac{Q}{U}
\]

where \( Q \) = volumetric flowrate of feed stream [m³/h].

The performance of a dissolved-air flotation system depends on having sufficient air bubbles to float all of the suspended solids. The relationship between the Air/Solid ratio, the solubility of air \( S_a \), the operating pressure \( P \), and the concentration of solids in the sludge is given below:

\[
\frac{A}{S} = \frac{1.3 \cdot S_a}{C_{solid}} \left( \frac{fP}{P_a} - 1 \right)
\]

where \( P_a = \) atmospheric pressure (1 atm).

In the case of operation with recycle, the corresponding equation is:
\[ \frac{A}{S} = \frac{1.3 \, S_a \, R}{C_{\text{solid}} \, Q} \left( \frac{f P}{P_a} - 1 \right) \]

where \( Q = \) volumetric flowrate of the feed stream \([m^3/h]\).

The mass flowrate of each component in the sludge stream is given by:

\[ S_i = F_i \, r_i \]

while the mass flowrate in the clarified stream is:

\[ C_i = F_i \, (1 - r_i) \]

Calculate surface area
If operation with recycle is specified
    If recycle ratio is specified
        If operating pressure is specified
            Calculate air-to-solids ratio
        else
            Calculate operating pressure
    else
        Calculate recycle ratio
else
    If operating pressure is specified
        Calculate air-to-solids ratio
    else
        Calculate operating pressure
Calculate air requirement
Do material balances
Adjust liquid content in sludge stream
Estimate cost

**Figure 7.3** Solution Algorithm for the Dissolved-Air Flotation Model
7.1.4 Granular Media Filtration

Granular media filtration is used for the supplemental removal of suspended solids from wastewater effluents of biological and chemical treatment processes. Removal is achieved by passing the wastewater through a (multi-layer) bed of granular material(s).

*Input Data:*

- Mass flowrate of component \(i\) in the feed stream, \(F_i\) [kg/h].
- Mass flowrate of component \(i\) in the wash stream, \(W_i\) [kg/h].
- Removal fraction of component \(i\), \(r_i\) [dimensionless].
- Superficial filtration velocity, \(V_s\) [m/h].
- Viscosity of the wastewater, \(\mu\) [cp].
- Density of the wastewater, \(\rho\) [kg/m\(^3\)].
- Number of layers, \(N\) [dimensionless].
- Height of each layer, \(L\) [m].
- Porosity of each layer, \(\alpha\) [dimensionless].
- Particle size of the medium in each layer, \(d\) [mm].
- Shape factor of the medium in each layer, \(\phi\) [dimensionless].
- Type of equation for the calculation of the clean bed headloss:
  
  (i) Carmen-Cozeny,
  
  (ii) Rose, or
  
  (iii) Fair-Hatch.
• Specification of:
  (i) Filtration time \(t\) [h], or
  (ii) Maximum allowable pressure drop through the filter \(\Delta P_{\text{max}}\) [bar].
• Maximum filter diameter \(D_{\text{max}}\) [m].
• Backwash time \(t_B\) [min].
• Backwash rate \(U_B\) [L/m\(^2\)-min].

\textit{Output Data:}
• Mass flowrate of component \(i\) in the filtered stream, \(C_i\) [kg/h].
• Mass flowrate of component \(i\) in the outlet wash stream, \(W_{Oi}\) [kg/h].
• Total clean bed headloss, \(H\) [m H\(_2\)O].
• Filter diameter, \(D\) [m].

If filtration time is specified:
• Overall pressure drop, \(\Delta P\) [bar].

If the maximum allowable pressure drop is specified:
• Filtration time, \(t\) [h].

\textit{Model:}
The cross sectional area of the filter \(A\) and its diameter \(D\) are calculated by:

\[
A = \frac{Q}{V_s} = \frac{\pi D^2}{4}
\]

where \(Q\) = volumetric flowrate of the feed stream [m\(^3\)/h].

To calculate the clean bed headloss \(H_j\) in the \(j^{th}\) layer of the filter (each layer is assumed to be uniform with respect to the particle size of the medium), any one of the following equations may be used:
Carmen-Cozeny

\[ H_j = \frac{f}{\phi} \frac{1 - \alpha}{\alpha^3} \frac{L}{d} \frac{V_s^2}{g} \]

\[ f = 150 \frac{1 - \alpha}{N_R} + 1.75 \]

Fair-Hatch

\[ H_j = k \nu S^2 \frac{f}{\phi} \frac{1 - \alpha}{\alpha^3} \frac{L}{d^2} \frac{V_s}{g} \]

Rose

\[ H_j = \frac{1.067}{\phi} C_D \frac{1}{\alpha^4} \frac{L}{d} \frac{V_s^2}{g} \]

\[ C_D = \frac{24}{N_R} + \frac{3}{N_R^{0.5}} + 0.34 \]

\[ N_R = \frac{\phi d V_s}{\nu} \]

where \( g = 9.81 \text{ m/s}^2 \).

\( f = \) friction factor [dimensionless].

\( k = \) filtration constant [dimensionless].

\( S = \) shape factor [dimensionless].

\( C_D = \) drag coefficient [dimensionless].
NR = Reynolds number [dimensionless].

ν = kinematic viscosity [m²/s].

The total clean bed headloss H is calculated by:

\[ H = \sum_{j} H_j \]

If the filtration time is specified, the head loss \( h_j \) in the \( j^{th} \) layer of the filter at time \( t \), is given by the equation (Sundstrom and Klei, 1979):

\[ h_j = H_j \left[ 1 + (2b+1) \frac{V}{\varepsilon} + (b+1)^2 \left( \frac{V}{\varepsilon} \right)^2 \right] \]

where \( b = \) packing constant, \( b = \frac{\varepsilon}{1-\varepsilon} \) [dimensionless].

\( \varepsilon = \) void fraction of bed [dimensionless].

\( V = \) volume of deposited particles per unit bed volume [dimensionless].

The total filter headloss \( h \) is calculated by:

\[ h = \sum_{j} h_j \]

while the equivalent overall pressure drop is given by:

\[ \Delta P = \rho g h \]

If \( \Delta P_{\text{max}} \) is specified, an iteration loop is used for the calculation of the filtration time \( t \). An initial \( t \) is guessed and \( \Delta P \) is calculated (following the above procedure) until the calculated \( \Delta P \) is within an acceptable limit from the specified \( \Delta P_{\text{max}} \). At this point the loop breaks and the recorded time is the filtration time \( t \).
The washwater requirement is:

\[ W_B = \frac{U_B A t_B}{1 + t_B} \]

The wash stream is adjusted by multiplication by the factor \( f \):

\[ f = \frac{W_B}{Q_B} \]

where \( Q_B \) = volumetric flowrate of the wash stream \([m^3/h]\).

The mass flowrate of each component in the filtered stream is given by:

\[ C_i = F_i (1 - r_i) \]

while the mass flowrate in the outlet wash stream is:

\[ WO_i = W_i + F_i r_i \]

---

**Figure 7.4 Solution Algorithm for the Granular Media Filtration Model**

- Calculate cross sectional area of filter
- Calculate filter diameter
- Calculate clean bed headloss
- If maximum allowable pressure drop is specified
  - Calculate filtration time
- else
  - Calculate pressure drop
- Calculate washwater requirement
- Adjust flowrate of wash stream
- Do material balances
- Estimate cost
7.1.5 Activated Carbon Adsorption

Adsorption is the process of collecting soluble substances that are in solution (adsorbate), on a suitable interface (adsorbent). Activated carbon is by far the most frequently used adsorbent. Activated carbon treatment of wastewater is a polishing process for water that has already received biological treatment. Activated carbon adsorption is also used in air-pollution applications for the control of VOCs.

**Input Data:**

- Mass flowrate of component \( i \) in the feed stream, \( F_i \) [kg/h].
- Mass flowrate of component \( i \) in the regeneration stream, \( R_i \) [kg/h].
- Removal fraction for each component, \( r_i \) [dimensionless].
- Viscosity of the entering fluid, \( \mu \) [cp].
- Carbon density, \( \rho_C \) [kg/m\(^3\)].
- Carbon particle size, \( d_C \) [mm].
- Binding capacity of the column, \( b \) [kg/kg carbon]
- Void fraction of the column, \( \varepsilon \) [dimensionless].
- Requirement of regeneration agent, \( S_r \) [kg/kg carbon].
- Specification of:
  (i) Linear velocity of the entering stream \( U \) [m/s], or
  (ii) Length to diameter ratio of the adsorption column, \( R \) [dimensionless].
• Breakthrough time, \( t_b \) [h].
• Regeneration time, \( t_r \) [h].

**Output Data:**
• Mass flowrate of component \( i \) in the cleaned stream, \( C_i \) [kg/h].
• Mass flowrate of component \( i \) in the exiting regeneration stream, \( R E_i \) [kg/h].
• Column length, \( L \) [m].
• Column diameter, \( D \) [m].

If length to diameter ratio, is specified:
• Linear velocity of the entering stream, \( U \) [m/s].

If linear velocity is specified:
• Length to diameter ratio of the column, \( R \) [dimensionless].

**Model:**
The quantity of adsorbate that can be taken up by an adsorbent in constant temperature, as a function of the adsorbate concentration, can be described by an adsorption isotherm. These isotherms apply for the adsorption of a single substance. Since most wastestreams contain more than one substance, the application of isotherms is limited. EnviroCAD instead, uses the concept of binding capacity, i.e., the maximum amount of adsorbate that can be adsorbed by the unit mass of the adsorbent.

The amount of material adsorbed is:

\[
m_d = \sum_i (F_i \rho_i)
\]

The carbon requirement is calculated by:
\[ m_C = \frac{m_g}{b} \]

The volume of carbon needed is:

\[ V_C = \frac{m_C}{\rho_C} \]

The volume of the column is then:

\[ V = \frac{V_C}{1 - \varepsilon} \]

and is related to the column geometric characteristics based on:

\[ V = A L = \frac{\pi D^2}{4} L = \frac{\pi D^3}{4} R \]

The linear velocity of the feed stream and the column diameter are related by:

\[ U = \frac{4 Q}{\pi D^2} \]

The regeneration agent requirement is:

\[ S = S_r m_C \]

The regeneration stream is adjusted by multiplication by the factor \( f \):

\[ f = \frac{S}{Q_B} \]
where $Q_B$ = volumetric flowrate of regeneration stream $[\text{m}^3/\text{h}]$.

The mass flowrate of each component in the cleaned stream is given by:

$$C_i = F_i (1 - r_i)$$

while the mass flowrate in the outlet wash stream is:

$$RE_i = R_i + F_i r_i$$

The solution algorithm for this model is shown in Figure 7.5.

- Calculate total amount of material adsorbed
- Calculate carbon requirement
- Calculate total carbon volume
- Calculate column volume
- If linear velocity is specified
  - Calculate column cross sectional area
  - Calculate column diameter
  - Calculate column length
  - Calculate length to diameter ratio
- else
  - Calculate column diameter
  - Calculate column length
  - Calculate column cross sectional area
  - Calculate linear velocity
- Calculate regeneration agent requirement
- Adjust regeneration stream flowrate
- Do material balances
- Estimate cost

Figure 7.5 Solution Algorithm for the Activated Carbon Adsorption Model
7.1.6 Air Stripping

Air stripping is used to treat wastewater which contains VOCs. During air stripping VOCs are transferred (stripped) from the wastewater to an air stream. This air stream may require further treatment to account for air pollution problems.

**Input Data:**

- Mass flowrate of component $i$ in the liquid feed stream, $L_i$ [kg/h].
- Mass flowrate of component $i$ in the gas feed stream, $G_i$ [kg/h].
- Fraction of component $i$ stripped, $r_i$ [dimensionless].
- Design component.
- Henry's law constant for the design component, $H$ [dimensionless].
- Characteristics of the packing material:
  - Packing constant, $C_f$ [dimensionless].
  - Total specific area per unit bed volume, $a_t$ [m$^2$/m$^3$].
  - Nominal diameter, $d_p$ [m].
  - Critical surface tension, $\sigma_C$ [dyn/cm].
  - Liquid surface tension, $\sigma_L$ [dyn/cm].
- Liquid phase density, $\rho_L$ [kg/m$^3$].
- Liquid phase viscosity, $\mu_L$ [cp].
- Gas phase viscosity, $\mu_G$ [cp].
• Diffusivity of design component in liquid phase, $D_L$ [cm$^2$/s].
• Diffusivity of design component in gas phase, $D_G$ [cm$^2$/s].
• Mode of operation:
  (i) Column diameter, $D$ [m].
  (ii) Overall pressure drop in the column, $\Delta P$ [Pa].
  (iii) Pressure drop per unit length of the column, $\frac{\Delta P}{Z}$ [Pa/m].

Output Data:
• Mass flowrate of component $i$ in the exiting liquid stream, $L_{E_i}$ [kg/h].
• Mass flowrate of component $i$ in the exiting gas stream, $G_{E_i}$ [kg/h].
• Height of the column, $Z$ [m].

If the column diameter has been specified:
• Overall pressure drop in the column, $\Delta P$ [Pa].
• Pressure drop per unit length of the column, $\frac{\Delta P}{Z}$ [Pa/m].

If the overall pressure drop in the column has been specified:
• Column diameter, $D$ [m].
• Pressure drop per unit length of the column, $\frac{\Delta P}{Z}$ [Pa/m].

If the pressure drop per unit length of the column has been specified:
• Column diameter, $D$ [m].
• Overall pressure drop in the column, $\Delta P$ [Pa].

Model:
EnviroCAD simulates a packed-bed stripping column. The two films theory is used for the sizing of the column. The assumption is that the liquid phase solution is dilute, with respect to the design component. The overall column height $Z$, is the product of the height of transfer unit $HTU$, times the number of transfer units $N_{OL}$:
\[ Z = HTU \cdot N_{OL} \]

The number of transfer units is given by:

\[
N_{OL} = \frac{S}{S-1} \ln \left[ \frac{(x_2 - \frac{y_1}{m})(S-1) + 1}{x_1 - \frac{y_1}{m}} \right]
\]

where \( S = \frac{mG}{L} \), stripping factor [dimensionless].

\( m \) = slope of the equilibrium curve (equal to the Henry's law constant for dilute solutions), [dimensionless].

\( y_1 \) = mole fraction of the entering gas, [dimensionless].

\( y_2 \) = mole fraction of the exiting gas, [dimensionless].

\( x_1 \) = mole fraction of the exiting liquid, [dimensionless].

\( x_2 \) = mole fraction of the entering liquid [dimensionless].

The height of each transfer unit is given by:

\[ HTU = \frac{L_m}{\rho L K_L a_w} \]

where \( L_m \) = superficial mass velocity of liquid, [kg/m²-h].

\( K_L \) = overall mass transfer coefficient for the liquid phase, [m/h].

\( a_w \) = wetted surface area of packing, [m²/m³].

The wetted area \( a_w \) is calculated from a correlation developed by Onda et al. (1968):

\[
\frac{a_w}{a_l} = 1 - \exp \left[ -1.45 \left( \frac{\sigma_C}{\sigma_L} \right)^{0.75} \left( Re_L \right)^{0.1} \left( Fr_L \right)^{-0.05} \left( We_L \right)^{0.2} \right]
\]
\[ \text{Re}_L = \frac{L_m}{a_t \mu_L}, \text{ Reynolds number} \]

\[ \text{Fr}_L = \frac{L^2_m}{\rho_L^2 g}, \text{ Froude number} \]

\[ \text{We}_L = \frac{L^2_m}{\rho_L \sigma_L a_t}, \text{ Weber number} \]

The overall mass transfer coefficient for the liquid phase \( K_L \) is given by:

\[ \frac{1}{K_L} = \frac{1}{k_G \text{m}} + \frac{1}{k_L} \]

where \( k_L \) = mass transfer coefficient for the liquid phase, [m/h].

\( k_G \) = mass transfer coefficient for the gas phase, [m/h].

\( k_L \) and \( k_G \) are calculated from the correlations developed by Onda et al. (1968):

\[ k_L \left( \frac{\rho_L}{\mu_L g} \right)^{1/3} = 0.0051 \left( \frac{L_m}{a_w \mu_L} \right)^{2/3} \left( \frac{\mu_L}{\rho_L D_L} \right)^{0.5} (a_t d_p)^{0.4} \]

\[ \frac{k_G}{a_t D_G} = 5.23 \left( \frac{G_m}{a_t \mu_G} \right)^{0.7} \left( \frac{\mu_G}{\rho_G D_G} \right)^{-1/3} (a_t d_p)^{-2} \]

where \( G_m = \frac{\Sigma G_i}{A} \), superficial mass velocity of gas [kg/m²-h].

The cross sectional area of the column and its diameter are related by:

\[ A = \frac{\pi D^2}{4} \]
The mass flowrate of each component in the exiting liquid stream is given by:

\[ LE_i = L_i (1 - r_i) \]

while the mass flowrate in the exiting gas stream is:

\[ GE_i = G_i + L_i r_i \]

The solution algorithm of the Air Stripping model (Figure 7.6) is based on Figure 6.34 of Treybal (1980). The curves of this figure have been correlated and introduced in the form of equations in EnviroCAD.

```
Do material balances
If column diameter is specified
    Calculate pressure drop per unit column height
    Calculate column height
    Calculate total pressure drop
else if pressure drop per unit column height is specified
    Calculate column diameter
    Calculate column height
    Calculate total pressure drop
else
    Calculate column diameter
    Calculate pressure drop per unit column height
    Calculate column height
Estimate cost
```

**Figure 7.6 Solution Algorithm for the Air Stripping Model**

### 7.2 Chemical Wastewater Treatment Unit Operations

Chemical processes are mainly used in wastewater treatment for neutralization of waste streams, for sludge conditioning, heavy metals removal by precipitation, and to enhance
sedimentation. Chemical treatment processes achieve their objectives by means of one or more chemical reactions. Their main characteristic is that an agent is added to the wastewater. All chemical processes that require addition of an agent, are simulated in EnviroCAD using the Neutralization model.

7.2.1 Neutralization

Neutralization is used for wastestreams that contain acidic or alkaline materials that require pH adjustment prior to discharge to receiving waters or prior to further treatment. For example, the pH in the biological systems should be maintained between 6.5 and 8.5 to ensure optimum biological activity (Eckenfelder, 1989).

Input Data:

- Mass flowrate of component i in the feed stream, $F_i$ [kg/h].
- Mass flowrate of neutralizing agent j in the neutralizing stream, $N_j$ [kg/h].
- Specification of neutralizing agent.
- Mass excess fraction of neutralizing agent, $E$ [dimensionless].
- Mass stoichiometric coefficient of component i in the neutralization reaction, $v_i$.
- Extent of reaction, $\xi$ [dimensionless].
- Liquid to total volume ratio, $R$ [dimensionless].
- Specific agitation power requirement, $P_s$ [kW/m$^3$].
- Specification of the operation mode:
(i) Batch, or
(ii) Continuous.

- If the batch mode is selected, the neutralizer liquid volume is:
  (i) Specified by the user, or
  (ii) Estimated by the system.

- If the continuous mode is selected, specification of:
  (i) Liquid volume, \( V \) [m\(^3\)], or
  (ii) Residence time, \( t \) [h].

**Output Data:**

- Mass flowrate of component \( i \) in the neutralized stream, \( C_i \) [kg/h].
- Total neutralizer volume, \( V_T \) [m\(^3\)].
- Total power requirement, \( P \) [kW].

If the batch mode is selected:

- If liquid volume is not specified by the user, neutralizer liquid volume \( V \) [m\(^3\)].

If the continuous mode is selected:

- If the residence time is specified, neutralizer liquid volume \( V \) [m\(^3\)].
- If the liquid volume is specified, residence time \( t \) [h].

**Model:**

For continuous operation, the residence time and liquid volume of the neutralizer are related by:

\[ t = \frac{V}{Q} \]

where \( Q = \) volumetric flowrate of feed stream [m\(^3\)/h].
For batch operation, when the neutralizer liquid volume is not specified by the user, it is set equal to the volume of material that is processed during each cycle.

The total neutralizer volume is:

\[ V_T = \frac{V}{R} \]

while the total power requirement is given by:

\[ P = V P_S \]

The mass of neutralizing agent required per unit mass reactant for complete reaction, \( r \), is calculated based on the reaction stoichiometry. The flowrate of the neutralizing stream is adjusted by multiplication by the factor \( f \):

\[ f = \frac{r m_r (1 + E)}{m_a} \]

where \( m_r \) = mass flowrate of the reactants in the feed stream [kg/h].

\( m_a \) = mass flowrate of the agent in the neutralizing stream [kg/h].

For the calculation of the material balances, the limiting component \( k \) is identified first based on the reaction stoichiometry and the composition of the feed stream. Its exiting mass flowrate \( C_k \) is:

\[ C_k = F_k (1 - \xi) \]

The exiting mass flowrates for the remaining components are:
\[ C_i = F_i (1 - \xi \frac{v_i}{v_k}) \]

where \( v_k \) = stoichiometric coefficient of the limiting component.

The solution algorithm for the neutralization model is shown in Figure 7.7.

```plaintext
If continuous mode of operation 
   If residence time is specified 
      Calculate liquid volume 
   else 
      Calculate residence time 
else 
   If liquid volume is not specified by the user 
      Calculate liquid volume 
Calculate total neutralizer volume 
Calculate total power requirement 
Calculate stoichiometric requirement for neutralizing agent 
Adjust neutralizing stream flowrate 
Do material balances 
Estimate cost
```

**Figure 7.7 Solution Algorithm for the Neutralization Model**

### 7.3 Biological Wastewater Treatment Unit Operations

Biological treatment processes constitute the heart of any pollution control flowsheet. The objectives of the biological treatment of wastewater are to coagulate and remove the nonsettleable colloidal solids and to stabilize the organic matter. With proper analysis and control, almost all wastewaters can be treated biologically. Biological treatment processes can be classified based on the state of the microorganisms responsible for the conversion of the organic matter, as suspended growth or attached growth.
7.3.1 Aeration Basin

The aeration basin is the first component of the activated sludge process, which is the principal suspended-growth biological treatment process. The organic waste introduced to the aeration basin is converted aerobically to new bacterial cells, CO$_2$, NH$_3$, H$_2$O and other end products. After a specified period of time, the aeration basin contents are passed into a settling tank, where the cells are separated from the treated wastewater.

**Input Data:**

- Mass flowrate of component $i$ in the feed stream, $F_i$ [kg/h].
- Mass flowrate of component $i$ in the inlet gas stream, $G_i$ [kg/h].
- Biodegradation fraction of component $i$, $b_i$ [dimensionless].
- Stripping fraction of component $i$, $v_i$ [dimensionless].
- Sorption fraction to biomass of component $i$, $s_i$ [dimensionless].
- Concentration of dissolved oxygen in the basin, DO [mg/L].
- Biomass yield, $Y$ [g biomass/g substrate biodegraded].
- Specific power requirement, $P_s$ [kW/m$^3$].
- Liquid to total volume ratio, $R$ [dimensionless].
- Specification of:
  (i) Hydraulic residence time $t$ [h], or
  (ii) Basin liquid volume $V$ [m$^3$].
Output Data:

- Mass flowrate of component i in the treated stream, \( T_i \) [kg/h].
- Mass flowrate of component i in the outlet gas stream, \( G_O_i \) [kg/h].
- Total basin volume, \( V_T \) [m³].
- Total power requirement, \( P \) [kW].

If hydraulic residence time has been specified:
- Basin liquid volume, \( V \) [m³].

If basin liquid volume has been specified:
- Hydraulic residence time, \( t \) [h].

Model:

The hydraulic residence time and liquid volume of the basin are related by:

\[
 t = \frac{V}{Q}
\]

where \( Q \) = volumetric flowrate of feed stream [m³/h].

The rate of oxygen flow \( r_0 \) required to maintain the concentration of dissolved oxygen in the tank in the specified level, is:

\[
 r_0 = \frac{V \cdot DO}{t}
\]

The flowrate of the inlet gas stream (which may be pure oxygen or air) is adjusted by multiplication by the factor \( f \):

\[
 f = \frac{r_0}{m_o}
\]
where \( m_0 \) = mass flowrate of oxygen in the inlet gas stream [kg/h].

The total basin volume is:

\[
V_T = \frac{V}{R}
\]

while the total power requirement is given by:

\[
P = V P_s
\]

The amount of biomass generated due to biodegradation, is:

\[
B = Y \sum_i (F_i b_i)
\]

The mass flowrate of each component in the exiting gas stream is given by:

\[
G_{Oi} = G_i + F_i v_i
\]

while the mass flowrate in the treated stream is:

\[
T_i = F_i (1 - b_i - v_i - s_i)
\]

The solution algorithm for the Aeration Basin model is shown in Figure 7.8.
If hydraulic residence time is specified
   Calculate basin liquid volume
else
   Calculate hydraulic residence time
Adjust flowrate of inlet gas stream
Calculate total basin volume
Calculate total power requirement
Do material balances
Calculate biomass generated
Estimate cost

**Figure 7.8** Solution Algorithm for the Aeration Basin Model

### 7.3.2 Trickling Filter

[Diagram of Trickling Filter]

A trickling filter is a bed of highly permeable medium to which microorganisms are attached and through which wastewater is trickled. The filter media usually are either rock or plastic. Trickling filters are the most frequently used attached-growth biological treatment processes.

**Input Data:**

- Mass flowrate of component i in the feed stream, \( F_i \) [kg/h].
- Biodegradation fraction of component i, \( b_i \) [dimensionless].
- Stripping fraction of component i, \( v_i \) [dimensionless].
- Hydraulic loading of the filter, \( H_L \) [m\(^3\)/m\(^2\)-h].
• Recycle ratio, \( \alpha \) [dimensionless].
• Operating temperature, \( T \) [°C].
• Biomass yield, \( Y \) [g biomass/g substrate biodegraded].
• Specification of sizing model:
  (i) NRC.
  (ii) Eckenfelder.
If NRC model specified:
• Weighing factor, \( \delta \) [dimensionless].
If Eckenfelder model specified:
• Constant \( n \) [dimensionless].
• Treatability factor at 20 °C, \( K_0 \) [1/d].
• Constant \( \theta \) [dimensionless].

Output Data:
• Mass flowrate of component \( i \) in the treated stream, \( T_i \) [kg/h].
• Mass flowrate of component \( i \) in the gas stream, \( G_i \) [kg/h].
• Filter volume, \( V \) [m\(^3\)].
• Filter depth, \( D \) [m].
• Cross sectional area of filter, \( A \) [m\(^2\)].
• Organic loading, \( OL \) [kg BOD\(_5\)/d].

Model:
The cross sectional area of the filter is calculated by:

\[
A = \frac{(1 + \alpha) Q}{HL}
\]
while the filter volume, depth, and cross sectional area are related by:

\[ V = AD \]

The total organic loading of the filter is:

\[ OL = Q S_i \]

where \( Q \) = volumetric flowrate of feed stream \([m^3/h]\).

\( S_i \) = BOD$_5$ concentration in the feed stream \([mg/l]\).

The amount of biomass generated due to biodegradation, is:

\[ B = Y \sum_i (F_i b_i) \]

The overall BOD$_5$ removal efficiency is:

\[ E = \frac{S_i - S_e}{S_i} \]

where \( S_e \) = BOD$_5$ concentration in the treated stream \([mg/L]\).

The NRC model (Water Environment Federation/American Society of Civil Engineers, 1982) is based on data from stone media trickling filters. It assumes that the contact between the filter media and the substrate depends on the filter dimensions. The filter volume is given by:

\[ V = \frac{0.000196 Q S_i}{R_f} \left[ \frac{E}{1 - E} \right]^2 \]
where \( R_f = \text{recycle factor}, \) defined as:

\[
R_f = \frac{1 + \alpha}{[1 + (1 - \delta) \alpha]^2}
\]

The Eckenfelder model (Water Environment Federation/American Society of Civil Engineers, 1982) assumes that the trickling filter can be represented as a plug flow reactor and the substrate degradation follows first order kinetics. The equation that relates the filter depth and its removal efficiency is:

\[
\frac{S_e (1 + \alpha)}{S_i + \alpha S_e} = \exp \left[ K \frac{D}{\left( \frac{A}{(1 + \alpha) Q} \right)^n} \right]
\]

where \( K = \text{treatability factor at operating temperature } T: \)

\[
K = K_0 e^{(T-20)}
\]

The mass flowrate of each component in the exiting gas stream is given by:

\[
G_i = F_i v_i
\]

while the mass flowrate in the treated stream is:

\[
T_i = F_i (1 - b_i - v_i)
\]

The assumption made in the above mass balance, is that removal by sorption to the solids is negligible, i.e. biodegradation and stripping are the only removal mechanisms.
Calculate cross sectional area of filter
Calculate total organic loading of the filter
Do material balances
Calculate biomass generated
Calculate overall BOD$_5$ removal efficiency
If NRC model is specified
  Calculate recycle factor
  Calculate filter volume
  Calculate filter depth
else
  Adjust treatability factor for temperature
  Calculate filter depth
  Calculate filter volume
Estimate cost

Figure 7.9 Solution Algorithm for the Trickling Filter Model

7.4 Sludge Treatment Unit Operations

Sludge is a “by-product” of the wastewater treatment. The sludge resulting from wastewater treatment processes is usually in the form of a liquid or semi-solid liquid that typically contains from 0.25 to 12% solids by weight (Metcalf & Eddy Inc., 1991). The sludge has to undergo a series of treatment steps before its final disposal. These treatment steps include thickening, digestion, dewatering and drying. The unit operations of EnviroCAD that simulate those processes, are presented in this section.

7.4.1 Gravity Thickening

![Gravity Thickening Diagram](Image)
Thickening is an operation used to increase the solids content of sludge by removing a portion of the liquid contained in the sludge, thus reducing the volume of sludge. Thickening is accomplished by gravity settling, flotation, centrifugation, and gravity belts (Metcalf & Eddy Inc., 1991). Gravity thickening is modeled in EnviroCAD.

**Input Data:**

- Mass flowrate of component i in the feed stream, \( F_i \) [kg/h].
- Fraction of component i transferred to sludge, \( r_i \) [dimensionless].
- Concentration of solid particles in sludge, \( X \) [mg/L].
- Terminal settling velocity of sludge, \( V_c \) [m/h]:
  - (i) Specified directly by the user, or
  - (ii) Estimated by the system.
- Specification of thickener geometry:
  - (i) Circular, or
  - (ii) Rectangular.

If the terminal settling velocity of the sludge is calculated by the system:

- Specification of settling velocity estimation model:
  - (i) Wahlberg and Keinath.
  - (ii) Daigger and Roper.
  - (ii) Generic.

If either one of the first two models is selected:

- Sludge volume index, \( SVI \) [mg/L].

If the generic model is selected:

- \( V_0 \) [m/h].
- \( A \) [mL/g].

If the rectangular thickener geometry is selected:
• Ratio of thickener length to thickener width, R [dimensionless].

Output Data:
• Mass flowrate each component in the clarified stream, $C_i$ [kg/h].
• Mass flowrate each component in the thickened stream, $T_i$ [kg/h].
• Surface area of the thickener, $A$ [m$^2$].

If the terminal settling velocity of the sludge is estimated by the system:
• Terminal settling velocity of the sludge, $V_c$ [m/h].

If circular thickener geometry has been specified:
• Thickener diameter $D$ [m].

If rectangular thickener geometry has been specified:
• Thickener length $L$ [m].
• Thickener width $W$ [m].

Model:
If the settling velocity is not specified directly, the settling flux theory is used to calculate it. The settling flux theory states that solids are transported to the bottom of the thickener by the settling velocity component and the velocity component due to withdrawal in the underflow, known as bulk flux. The settling flux equation used in the model for the calculation of the settling velocity is of the generic form:

$$V_S = V_0 \exp(-KX)$$

where $V_S$ = sludge interface settling velocity
$X$ = concentration of suspended solids
$V_0$ = adjustable parameter
If the sludge volume index is known, the settling velocity can be calculated using one of the following two equations. The first relationship (Daigger and Roper, 1985) is:

\[ V_s = 7.80 \exp[-(0.148 + 0.0021 \text{ SVI}) X] \]

where \( 35 \text{ mL/g} < \text{SVI} < 403 \text{ mL/g} \).

The second relationship, by Whalberg and Keinath (1988), is:

\[ V_s = (15.3 - 0.0615 \text{ SVI}) \exp[-(0.426 + 0.00384 \text{ SVI} - 0.0000543 \text{ SVI}^2) X] \]

where \( 34 \text{ mL/g} < \text{SVI} < 219 \text{ mL/g} \).

Once the settling velocity \( V_s \) is known, the sedimentation area \( A \) can be calculated:

\[ A = \frac{Q}{V_s} \]

The area \( A \), and the geometrical characteristics of the thickener, are related by:

\[ A = \frac{\pi D^2}{4} = L W = R W^2 \]

The mass flowrate of each component in the thickened stream is given by:

\[ T_i = F_i r_i \]

while the mass flowrate in the clarified stream is:
\[ C_i = F_i (1 - r_i) \]

The solution algorithm for the Gravity Thickening model is given in Figure 7.10.

If settling flux is specified by the user
   Calculate surface area of thickener
else
   Calculate settling flux
   Calculate surface area of thickener
If circular thickener geometry is specified
   Calculate thickener diameter
else
   Calculate thickener length
   Calculate thickener width
Do material balances
Estimate cost

**Figure 7.10 Solution Algorithm for the Gravity Thickening Model**

### 7.4.2 Anaerobic Digestion

![Diagram of Anaerobic Digestion](image)

Anaerobic digestion is one of the oldest processes used for the stabilization of sludges. It involves the decomposition of organic and inorganic matter in the absence of molecular oxygen. The influent to this process may be sludge from primary sedimentation, biological sludge or a mixture of both.
**Input Data:**

- Mass flowrate of component i in the feed stream, $F_i$ [kg/h].
- Mass stoichiometric coefficient of component i in the digestion reaction, $v_i$.
- Fraction of component i transferred to the digested sludge, $s_i$ [dimensionless].
- Extent of reaction, $\xi$ [dimensionless].
- Identification of the reaction gases.
- Solids concentration in the output (digested) sludge stream, $C_{\text{solid}}$ [mg/L].
- Specification of the digester type:
  - (i) Standard-rate: The digester contents are unheated and unmixed.
  - (ii) High-rate: The contents are heated and completely mixed.
- Specification of:
  - (i) Active digester volume $V_a$ [m$^3$], or
  - (ii) Solids retention time in the digester, SRT [d].
- Ratio of active to total digester volume, $R$ [dimensionless].
  
  If a high-rate digester is selected:
- Digestion temperature $T$ [°C].
- Heating value of the methane generated by the digestion, $H_{\text{methane}}$ [kcal/m$^3$].
- Efficiency of the heating units, $E$ [dimensionless].
- Specific power required for mixing, $P$ [kW/m$^3$].

**Output Data:**

- Mass flowrate of component i in the gas stream, $G_i$ [kg/h].
- Mass flowrate of component i in the liquid stream, $L_i$ [kg/h].
- Mass flowrate of component i in the sludge stream, $S_i$ [kg/h].
- Total digester volume $V$ [m$^3$].
- Total gas production from the digestion $P_{\text{methane}}$ [m$^3$/h].
If a high-rate digester is selected:

- Digestion gas required for heating, \( Q_{\text{methane}} \, [m^3/h] \).
- Heating duty required, in addition to the combustion of digestion gas, \( H \, [\text{kcal/h}] \).
- Natural gas required for the additional heating, \( Q_{\text{gas}} \, [m^3/h] \).

**Model:**

The biological conversion of the organic matter in the anaerobic digestion process occurs in three steps, which are presented in the following paragraphs that are adapted from Metcalf & Eddy Inc. (1991), and Grady and Lim (1980).

The first step (hydrolysis), involves the enzymatic transformation of higher molecular weight compounds into compounds suitable for use as a source of energy and cell carbon.

\[
\text{Insoluble Organics, Large Soluble Organics} \rightarrow \text{Soluble Organic Matter}
\]

In the second step (acidogenesis), the small soluble molecules resulting from hydrolysis are used as carbon and energy sources by bacteria which carry out biochemical reactions. The end products of those reactions are short-chain acids (e.g., formic acid, acetic acid), carbon dioxide and hydrogen.

\[
\text{Soluble Organic Matter} \rightarrow \text{Short-chain Acids} + \text{Carbon Dioxide} + \text{Hydrogen}
\]

In the third step (methanogenesis), the products of the second step (formic acid, acetic acid, carbon dioxide and hydrogen) are utilized by methanogenic bacteria (which are strict anaerobes) to produce methane gas.
$$4\text{H}_2 + \text{CO}_2 \rightarrow \text{CH}_4 + 2\text{H}_2\text{O}$$
$$\text{CH}_3\text{COOH} \rightarrow \text{CH}_4 + \text{CO}_2$$
$$4\text{HCOOH} \rightarrow \text{CH}_4 + 3\text{CO}_2 + 2\text{H}_2\text{O}$$

The active digester volume and the solids residence time are related by:

$$V_a = Q \text{ SRT}$$

where $Q =$ volumetric flowrate of feed stream [m$^3$/h].

The total digester volume is given by:

$$V = \frac{V_a}{R}$$

For the calculation of the material balances, the limiting component $k$ is identified first based on the reaction stoichiometry and the composition of the feed stream. Its exiting mass flowrate $L_k$ is:

$$L_k = F_k (1 - \xi)$$

The exiting mass flowrates for the remaining components are:

$$L_i = F_i (1 - \xi \frac{v_i}{v_k})$$

where $v_k =$ stoichiometric coefficient of the limiting component.

If a component has been specified as a digestion gas, $G_i = L_i$. The component mass flowrates in the digested sludge stream are calculated as:
\[ S_i = L_i s_i \]

The combustion of the digestion gas generated, can be used as a heat source. If the high-rate digester type has been selected, the amount of digestion gas required for the heating of the sludge mixture is calculated. This amount is calculated based on the heating value of the digestion gas. If the amount of the digestion gas produced is not enough, combustion of natural gas is assumed to provide the remaining heating duty. The heating value of the natural gas is taken as 8900 kcal/m^3 (Qasim, 1994).

The solution algorithm for the anaerobic digestion model is shown in Figure 7.11.

```plaintext
If active digester volume is specified
    Calculate solids residence time
else
    Calculate active digester volume
Calculate total digester volume
Do reaction calculations
Do material balances
Adjust liquid content in sludge stream
Calculate volume of methane produced
If high rate digester is specified
    Calculate thermal duty for heating of the sludge
    Calculate equivalent methane required for sludge heating
    If the generated methane is insufficient for heating
        Calculate additional natural gas needed
    Calculate electrical power required for mixing
Estimate cost
```

Figure 7.11 Solution Algorithm for the Anaerobic Digestion Model
7.4.3 Belt Filter Press

Belt filter presses are continuous-feed sludge dewatering devices that involve the application of chemical conditioning, gravity drainage, and mechanically applied pressure to dewater sludge (Metcalf & Eddy Inc., 1991). The objective is to achieve further reduction of the volume of the sludge.

**Input Data:**

- Mass flowrate of component i in the feed stream, $F_i$ [kg/h].
- Mass flowrate of component i in the washwater stream, $W_i$ [kg/h].
- Fraction of component i transferred to the sludge cake, $r_i$ [dimensionless].
- Specific washwater requirement, $W_s$ [m$^3$/m-h].
- Solids content in the sludge cake, $C_{solid}$ [wt. %].
- Specification of:
  - (i) Solids loading rate $L$ [kg/m-h], or
  - (ii) Belt width, $W$ [m].

**Output Data:**

- Mass flowrate of component i in the sludge cake stream, $S_i$ [kg/h].
- Mass flowrate of component i in the exiting washwater stream, $WO_i$ [kg/h].

If the solids loading rate is specified:

- Belt width, $W$ [m].

If the belt width is specified:
• Solids loading rate, \( L \) [kg/m-h].

*Model:*

The solids loading and the belt width are related by:

\[
W = \frac{S}{L}
\]

where \( S \) = mass flowrate of solids in the feed stream [kg/h].

The washwater requirement is:

\[
R = W S W
\]

The wash stream is adjusted by multiplication by the factor \( f \):

\[
f = \frac{R}{Q}
\]

where \( Q \) = volumetric flowrate of the washwater stream [m\(^3\)/h].

The mass flowrate of each component in the sludge cake stream is given by:

\[
S_i = F_i r_i
\]

while the mass flowrate in the outlet wash stream is:

\[
WO_i = W_i + F_i (1 - r_i)
\]

The solution algorithm for this Belt Filter Press model is given in Figure 7.12.
Calculate amount of solids in the feed stream
If belt width is specified
    Calculate solids loading rate
else
    Calculate belt width
Calculate washwater requirement
Adjust washwater flowrate
Do material balances
Adjust liquid content in sludge cake
Calculate total solids recovery
Estimate cost

**Figure 7.12** Solution Algorithm for the Belt Filter Press Model

### 7.4.4 Heat Drying

Heat drying of the sludge accomplishes the reduction of the water content of the sludge, by vaporization of water to the air (Metcalf & Eddy Inc., 1991). The objective of heat drying is to reduce the water content of the sludge to less than 10%. Rotary and spray dryers are among the most common units used for drying sludge thermally.

#### 7.4.4.1 Rotary Dryer

![Rotary Dryer Diagram]

A rotary dryer consists of a revolving cylinder, horizontal or slightly inclined toward the outlet. Wet feed enters one end of the cylinder and dry material discharges from the other. Heating in rotary dryers usually is provided by direct contact of the hot gas with the wet material, but indirect heating (hot gas passing through an external jacket) may also be used.
**Input Data:**

- Mass flowrate of component i in the feed stream, $F_i$ [kg/h].
- Mass flowrate of component i in the drying gas stream, $G_i$ [kg/h].
- Evaporation fraction of component i, $r_i$ [dimensionless].
- Length to diameter ratio of the dryer, $R$ [dimensionless].
- Specific heating per unit solvent (water) evaporated, $H_s$ [kJ/kg solvent].
- Specific power per unit peripheral drum area, $P_s$ [kW/m$^2$].
- Specification of dryer type:
  (i) Direct, or
  (ii) Indirect.
- Specification of the operation mode:
  (i) Design, or
  (ii) Rating.

If direct type dryer is selected:

- Drying gas requirement per unit cross-sectional drum area, $R_s$ [kg dry gas/m$^2$-s].

If the design mode is selected:

- Overall evaporation rate per unit dryer volume, $E$ [kg solvent/m$^3$-h].
- Maximum dryer diameter, $D_{\text{max}}$ [m].

If the rating mode is selected:

- Dryer diameter, $D$ [m].
- Number of units, $N$ [dimensionless].

**Output Data:**

- Mass flowrate of component i in the dried stream, $D_i$ [kg/h].
- Mass flowrate of component i in the drying gas outlet stream, $G_{O_i}$ [kg/h].
- Total heating duty, $H$ [kcal/h].
• Total power, \( P \) [kW].
• Dryer length, \( L \) [m].

If the design mode is selected:
• Dryer diameter, \( D \) [m].
• Number of units \( N \).

If the rating mode is selected:
• Overall evaporation rate per unit dryer volume, \( E \) [kg solvent/m\(^3\)-h].

_Model:

The total amount of water removed is:

\[
m_w = \sum_i (F_i r_i)
\]

The dryer volume is related to its geometrical characteristics by:

\[
V = \frac{\pi D^2}{4}, \quad L = \frac{\pi D^3}{4}, \quad R
\]

The number of unit is given by:

\[
N = \frac{V_{\text{max}}}{V}
\]

The evaporation rate is calculated by:

\[
E = \frac{m_w}{V N}
\]

The total drying gas requirement is:
\[ m_g = \frac{\pi D^2}{4} R \]

and so the drying gas stream is multiplied by the adjustment factor \( f \):

\[ f = \frac{m_g}{\sum_i G_i} \]

The total heating duty \( H \) is given by:

\[ H = m_w H_S \]

while the total power requirement is calculated by:

\[ P = (\pi D L) P_S \]

The mass flowrate of each component in the dried stream is given by:

\[ D_i = F_i (1 - r_i) \]

while the mass flowrate in the outlet drying gas stream is:

\[ G_{O_i} = G_i + F_i r_i \]

The solution algorithm for the Rotary Dryer model is given in Figure 7.13.
Calculate total water removed
If rating mode is specified
   Calculate dryer volume
   Calculate evaporation rate
else
   Calculate maximum dryer volume
   Calculate dryer volume
   Calculate number of units
   Calculate dryer diameter
Calculate dryer length
Calculate drying air requirement
Adjust drying air stream
Calculate heating duty
Calculate power requirement
Do material balances
Estimate cost

**Figure 7.13** Solution Algorithm for the Rotary Dryer Model

### 7.4.4.2 Spray Dryer

![Spray Dryer Diagram](image)

In a spray dryer liquid sludge is dispersed into a stream of hot gas in the form of a mist or fine droplets. Moisture is rapidly vaporized from the droplets, leaving residual particles of dry sludge, which are then separated from the gas stream.

**Input Data:**
- Mass flowrate of component $i$ in the feed stream, $F_i$ [kg/h].
- Mass flowrate of component $i$ in the drying gas stream, $G_i$ [kg/h].
• Evaporation fraction of component \( i \), \( r_i \) [dimensionless].

• Height to diameter ratio of the dryer, \( R \) [dimensionless].

• Specific heating per unit solvent (water) evaporated, \( H_S \) [kJ/kg solvent].

• Specific power per unit feed volume, \( P_S \) [kW/m\(^3\)].

• Drying gas requirement per unit solvent evaporated, \( R_S \) [kg dry gas/kg solvent].

• Specification of the operation mode:
  
  (i) Design.
  
  (ii) Rating.

If the design mode is selected:

• Overall evaporation rate per unit dryer volume, \( E \) [kg solvent/m\(^3\)-h].

• Maximum dryer diameter, \( D_{\text{max}} \) [m].

If the rating mode is selected:

• Dryer diameter, \( D \) [m].

• Number of units, \( N \) [dimensionless].

\textit{Output Data:}

• Mass flowrate each component in the dried stream, \( D_i \) [kg/h].

• Mass flowrate each component in the drying gas outlet stream, \( G_{Oi} \) [kg/h].

• Total heating duty, \( H \) [kcal/h].

• Total power, \( P \) [kW].

• Dryer Height, \( L \) [m].

If the design mode is selected:

• Dryer diameter, \( D \) [m].

• Number of units \( N \).

If the rating mode is selected:

• Overall evaporation rate per unit dryer volume, \( E \) [kg solvent/m\(^3\)-h].
**Model:**

The total amount of water removed is:

\[ m_w = \sum_i (F_i r_i) \]

The total drying gas requirement is:

\[ m_g = m_w R_S \]

and so the drying gas stream is multiplied by the adjustment factor \( f \):

\[ f = \frac{m_g}{\sum_i G_i} \]

The dryer volume is related to its geometrical characteristics by:

\[ V = \frac{\pi D^2}{4} \quad L = \frac{\pi D^3}{4} \quad R \]

The number of unit is given by:

\[ N = \frac{V_{max}}{V} \]

The evaporation rate is calculated by:

\[ E = \frac{m_w}{V N} \]

The total heating duty \( H \) is given by:
\[ H = m_w H_S \]

while the total power requirement is calculated by:

\[ P = Q P_S \]

where \( Q = \) volumetric flowrate of feed stream \([m^3/h]\).

The mass flowrate of each component in the dried stream is given by:

\[ D_i = F_i (1 - r_i) \]

while the mass flowrate in the outlet drying gas stream is:

\[ G_{Oi} = G_i + F_i r_i \]

---

Calculate total water removed  
Calculate drying air requirement  
Adjust drying air stream  
If rating mode is specified  
   Calculate dryer volume  
   Calculate evaporation rate  
else  
   Calculate maximum dryer volume  
   Calculate dryer volume  
   Calculate number of units  
   Calculate dryer diameter  
Calculate dryer height  
Do material balances  
Estimate cost

---

**Figure 7.14** Solution Algorithm for the Spray Dryer Model
7.5 Air Pollution Control Unit Operations

Air pollution control units are used to minimize the release to the atmosphere of the five primary criteria pollutants (Cooper and Alley, 1994) and of VOCs, which because of their widespread use are recognized as another major air pollutant.

7.5.1 Gas Absorption

Gas absorption refers to the selective transfer of one or more gaseous components, through a gas-liquid interface, to a contacting liquid. Applications of gas absorption to air pollution control include the removal of pollutants from gas streams, and the recovery of precious gaseous components (e.g. organic solvents).

**Input Data:**

- Mass flowrate of component i in the liquid feed stream, \(L_i\) [kg/h].
- Mass flowrate of component i in the gas feed stream, \(G_i\) [kg/h].
- Fraction of component i absorbed, \(r_i\) [dimensionless].
- Design component.
- Henry's law constant for the design component, \(H\) [dimensionless].
- Characteristics of the packing material:
  - Packing constant, \(C_f\) [dimensionless].
  - Total specific area per unit bed volume, \(a_t\) [m²/m³].
Nominal diameter, \(d_p\) [m].

Critical surface tension, \(\sigma_C\) [dyn/cm].

Liquid surface tension, \(\sigma_L\) [dyn/cm].

- Liquid phase viscosity, \(\mu_L\) [cp].
- Gas phase density, \(\rho_G\) [kg/m\(^3\)].
- Gas phase viscosity, \(\mu_G\) [cp].
- Diffusivity of design component in liquid phase, \(D_L\) [cm\(^2\)/s].
- Diffusivity of design component in gas phase, \(D_G\) [cm\(^2\)/s].
- Specification of either one of the following:
  
  (i) Column diameter, \(D\) [m].
  
  (ii) Overall pressure drop in the column, \(\Delta P\) [Pa].
  
  (iii) Pressure drop per unit length of the column, \(\frac{\Delta P}{Z}\) [Pa/m].

**Output Data:**

- Mass flowrate of component \(i\) in the exiting liquid stream, \(L_{E_i}\) [kg/h].
- Mass flowrate of component \(i\) in the exiting gas stream, \(G_{E_i}\) [kg/h].
- Height of the column, \(Z\) [m].

If the column diameter has been specified:

- Overall pressure drop in the column, \(\Delta P\) [Pa].
- Pressure drop per unit length of the column, \(\frac{\Delta P}{Z}\) [Pa/m].

If the overall pressure drop in the column has been specified:

- Column diameter, \(D\) [m].
- Pressure drop per unit length of the column, \(\frac{\Delta P}{Z}\) [Pa/m].

If the pressure drop per unit length of the column has been specified:

- Column diameter, \(D\) [m].
- Overall pressure drop in the column, \(\Delta P\) [Pa].
Model:

EnviroCAD simulates a packed-bed absorption column. The two films theory is used for the sizing of the column. The assumption is that the gas phase solution is dilute, with respect to the design component. The overall column height \( Z \), is the product of the height of transfer unit \( HTU \), times the number of transfer units \( N_{OG} \):

\[
Z = HTU \times N_{OG}
\]

The number of transfer units is given by:

\[
N_{OG} = \frac{A}{A-1} \ln \left[ \frac{(y_1 - mx_2)(A-1) + 1}{y_2 - mx_2 A} \right]
\]

where \( A = \frac{L}{mg} \), absorption factor [dimensionless].

\( m \) = slope of the equilibrium curve (equal to the Henry's law constant for dilute solutions) [dimensionless].

\( y_1 \) = mole fraction of the entering gas [dimensionless].

\( y_2 \) = mole fraction of the exiting gas [dimensionless].

\( x_1 \) = mole fraction of the exiting liquid [dimensionless].

\( x_2 \) = mole fraction of the entering liquid [dimensionless].

The height of each transfer unit is given by:

\[
HTU = \frac{G_m}{\rho_g K_G a_w}
\]

where \( G_m \) = superficial mass velocity of gas [kg/m\(^2\)-h].

\( K_G \) = overall mass transfer coefficient for the gas phase [m/h].
\( a_w = \) wetted area of packing \([\text{m}^2/\text{m}^3]\).

\( a_w \) is calculated from the Onda correlation presented in section 7.1.6. The overall mass transfer coefficient for the gas phase \( K_G \) is given by:

\[
\frac{1}{K_G} = \frac{1}{k_G} + \frac{m}{k_L}
\]

where \( k_L \) = mass transfer coefficient for the liquid phase \([\text{m}/\text{h}]\).

\( k_G \) = mass transfer coefficient for the gas phase \([\text{m}/\text{h}]\).

\( k_L \) and \( k_G \) are calculated from the correlations presented in section 7.1.6. The cross sectional area of the column and its diameter are related by:

\[
A = \frac{\pi D^2}{4}
\]

The mass flowrate of each component in the exiting gas stream is given by:

\[
G_{E_i} = G_i (1 - r_i)
\]

while the mass flowrate in the exiting liquid stream is:

\[
L_{E_i} = L_i + G_i r_i
\]

The solution algorithm of the Absorption model (Figure 7.15) is based on Figure 6.34 of Treybal (1980). The curves of this figure have been correlated and introduced in the form of equations in EnviroCAD.
Figure 7.15 Solution Algorithm for the Absorption Model

7.5.2 Cyclone

Cyclone collectors are widely used for the removal of particulate matter from process gases. Cyclones alone are not generally adequate to meet stringent air pollution regulations but they are ideal as pre-cleaners for more expensive devices such as baghouse filters and electrostatic precipitators.

Input Data:

- Mass flowrate of component i in the feed stream, $F_i$ [kg/h].
- Superficial velocity of the feed gas stream, $U$ [m/s].
• Maximum allowable pressure drop, $\Delta P_{\text{max}}$ [bar].
• Collection efficiency of component $i$, $\eta_i$ [dimensionless]:
  (i) Specified directly by the user, or
  (ii) Estimated by the system.
If the collection efficiency is estimated by the system:
• Mean particle diameter in size range $i$, $d_i$ [$\mu$m].
• Mass fraction of particles in size range $i$, $m_i$ [dimensionless].
• Mean density of particles, $\rho_p$ [kg/m$^3$].
• Density of inlet gas, $\rho$ [kg/m$^3$].
• Viscosity of inlet gas, $\mu$ [cp].

*Output Data:*
• Mass flowrate of component $i$ in the clean stream, $C_i$ [kg/h].
• Mass flowrate of component $i$ in the particulate stream, $P_i$ [kg/h].
• Pressure drop, $\Delta P$ [bar].
• Inlet duct height, $a$ [m].
• Inlet duct width, $b$ [m].
• Body diameter, $D$ [m].
• Gas outlet diameter, $D_e$ [m].
• Height of upper cylindrical body, $h$ [m].
• Outlet length, $S$ [m].
• Overall cyclone height, $H$ [m].
• Outlet duct diameter for dust, $B$ [m].

If the collection efficiency is estimated by the system:
• Overall collection efficiency $\eta$ [dimensionless].
Model:

The cross sectional area of the rectangular inlet duct is given by:

\[ A = \frac{Q}{U} \]

where \( Q \) = volumetric flowrate of feed stream \([m^3/h]\).

The geometrical characteristics of the inlet duct are related to the body diameter \( D \), based on the proportions proposed by Lapple and Sepherd (1939):

Inlet duct height, \( a = \frac{D}{2} \)

Inlet duct width, \( b = \frac{D}{4} \)

Therefore the cross sectional area of the inlet duct is related to the cyclone body by:

\[ A = a \times b = \frac{D}{2} \times \frac{D}{4} = \frac{D^2}{8} \]

The remaining design characteristics of the cyclone are also related to \( D \):

Gas outlet diameter, \( D_e = \frac{D}{4} \)

Height of upper cylindrical body, \( h = 2D \)

Outlet length, \( S = \frac{5D}{8} \)

Overall cyclone height, \( H = 4D \)
Outlet duct diameter for dust, \( B = \frac{D}{4} \)

For the calculation of pressure drop the approach proposed by Shepherd and Lapple (1939) is used. The pressure drop is expressed in number of inlet velocity heads \( H_v \):

\[
H_v = K \left( \frac{\frac{a b}{D^2}}{D_e} \right)
\]

The constant \( K \), depends on cyclone configuration and operating conditions (\( K = 16 \) usually). The number of inlet velocity heads can be converted to a static pressure drop:

\[
\Delta P = \frac{1}{2} \rho U^2 H_v
\]

The removal efficiency \( \eta_i \) for each particle size, if it is not specified directly, is calculated by the system based on the model proposed by Leith and Licht (1972):

\[
\eta_i = 100 \left( 1 - \exp\left[ -2(C\Psi)^{1/(2n+2)} \right] \right)
\]

\( C \) is a dimensionless design number, which depends on the physical shape of the cyclone. For a cyclone with rectangular inlet duct, \( C \) is defined as:

\[
C = \frac{8K_c}{K_aK_b}
\]

\[
K_a = \frac{a}{D}
\]

\[
K_b = \frac{b}{D}
\]
\[ K_c = \frac{V_s + \frac{V_{nl}}{2}}{D^3} \quad \text{or} \quad K_c = \frac{V_s + \frac{V_H}{2}}{D^3} \]

\[ V_s = \frac{\pi (S - \frac{a}{2})(D^2 - D_e^2)}{4} \]

The calculation of \( V_{nl} \) or \( V_H \) is based on the natural length of the cyclone, given by:

\[ l = 2.3 D_e \left( \frac{D_e^2}{ab} \right)^{1/3} \]

If \( l < (H - S) \), then calculate \( V_{nl} \):

\[ V_{nl} = \frac{\pi D^2}{4} (h - S) + \frac{\pi D^2}{4} \left( \frac{l + S - h}{3} \right) \left( 1 + \frac{d}{D} + \frac{d^2}{D^2} \right) - \frac{\pi D_e^2 l}{4} \]

\[ d = D - (D - B) \left( \frac{S + 1 - h}{H - h} \right) \]

If \( l > (H - S) \), then calculate \( V_H \):

\[ V_H = \frac{\pi D^2}{4} (h - S) + \frac{\pi D^2}{4} \left( \frac{H - h}{3} \right) \left( 1 + \frac{B}{D} + \frac{B^2}{D^2} \right) - \frac{\pi D_e^2}{4} (H - S) \]

\( \Psi \) is a dimensionless impaction parameter depending upon the operating conditions:

\[ \Psi = \frac{\rho_p d_p^2 U}{18 \mu D} (n + 1) \]

where \( n = \) dimensionless exponent of the vortex law for tangential velocity distribution.
The overall collection efficiency $\eta$ is:

$$\eta = \sum_{i} (m_i \eta_i)$$

The mass flowrate of each component in the particulate stream is given by:

$$P_i = F_i r_i$$

while the mass flowrate in the clean stream is:

$$C_i = F_i (1 - r_i)$$

While pressure drop greater than maximum allowable
- Calculate cross sectional area of inlet duct
- Calculate body diameter
- Calculate inlet duct height
- Calculate inlet duct width
- Calculate outlet gas diameter
- Calculate height of upper cylindrical body
- Calculate outlet duct length
- Calculate outlet duct diameter
- Calculate overall cyclone height
- Calculate pressure drop

If removal efficiency is calculated by the system
- Calculate $K_a$
- Calculate $K_b$
- Calculate $K_c$
- Calculate $C$
- Calculate impaction parameter $\Psi$
- Calculate removal efficiency

Do mass balances
Estimate cost

Figure 7.16 Solution Algorithm for the Cyclone Model
7.5.3 Fabric Filtration

Fabric filtration is a method for separating dry particles from a gas stream. In fabric filtration, the dusty gas flows into and through a number of filter bags placed in parallel (baghouse), leaving the dust retained by the fabric (Cooper and Alley, 1994). The collected dust must be periodically removed from the baghouse.

**Input Data:**
- Mass flowrate of component i in the feed stream, $F_i$ [kg/h].
- Removal efficiency of component i, $r_i$.
- Superficial filtering velocity (air/cloth ratio), $v$ [m/min].
- Number of compartments, $N$ [dimensionless].
- Cleaning time for one compartment, $t_c$ [min].
- The parameters of the filter drag model:
  - Effective drag coefficient, $K_e$ [Pa-min/m].
  - Specific drag coefficient, $K_s$ [Pa-min-m/kg].
- For one compartment, specification of either:
  (i) Maximum allowable pressure drop $\Delta P_{max}$ [Pa], or
  (ii) Filtration time $t_f$ [min].

**Output Data:**
- Mass flowrate of component i in the clean stream, $C_i$ [kg/h].
- Mass flowrate of component i in the dust stream, $D_i$ [kg/h].
• Total filtration area, $A \text{ [m}^2\text{]}$.

If the maximum allowable pressure drop is specified:

• Filtration time, $t_f \text{ [min]}$.

If the filtration time is specified:

• Pressure drop, $\Delta P \text{ [Pa]}$.

Model:

The dust loading $L \text{ [kg/m}^3\text{]}$ is given by:

$$L = \frac{\sum F_i}{Q}$$

where $Q = \text{volumetric flowrate of feed stream [m}^3\text{/h]}$.

The total filtration area $A$, is calculated by:

$$A = \frac{Q}{v}$$

while the filtration area per compartment $A_c$ is:

$$A_c = \frac{A}{v}$$

The time between the cleanings of any two compartments is defined as the filter run time $t_r$. The filtration time $t_f$ is related to $t_r$ and $t_c$ by:

$$t_f = N(t_r + t_c) - t_c$$

During the time $t_j$, the cloth in compartment $j$ accumulates an areal dust density $W_j$: 
\[ W_j = (N - 1) \left( V_N \, L \, t_r + V_{N-1} \, L \, t_c \right) \]

\[ V_N = \frac{Q}{N \, A_c} \], filtering velocity when all compartments are filtering

\[ V_{N-1} = \frac{Q}{(N - 1) \, A_c} \], filtering velocity when one compartment is off-line for cleaning

The filter drag in compartment \( j \), \( S_j \), is given by:

\[ S_j = K_e + K_s \, W_j \]

The actual filtering velocity \( V_j \) in compartment \( j \) at time \( t_j \), is given by:

\[ V_j = f_N \, V_{N-1} \]

The correction factor \( f_N \) is a function of the number of compartments \( N \) (Cooper and Alley, 1994):

\[ \ln(f_N) = 0.24488 - 0.41681 \ln N + 0.058905 (\ln N)^2 \]

Finally, the pressure drop \( \Delta P_j \) for compartment \( j \) is:

\[ \Delta P_j = S_j \, V_j \]

The mass flowrate of each component in the particulate stream is given by:

\[ P_i = F_i \, r_i \]
while the mass flowrate in the clean stream is:

\[ C_i = F_i (1 - r_i) \]

The solution algorithm for the Fabric Filtration model is given in Figure 7.17.

```
Calculate dust loading
Estimate correction factor
Calculate actual filtering velocity
If maximum allowable pressure drop is specified
    Calculate filter drag
    Calculate dust density
    Calculate run time
    Calculate filtration time
else
    Calculate run time
    Calculate dust density
    Calculate filter drag
    Calculate pressure drop
Do material balances
Estimate cost
```

**Figure 7.17** Solution Algorithm for the Fabric Filtration Model

### 7.5.4 Electrostatic Precipitation

![Electrostatic Precipitation Diagram]

Feed Gas
   → Clean Gas
   → Collected Dust
Electrostatic precipitators remove suspended particles from gases by exposing them to a high-voltage electric field. As the particles pass through the precipitator, they become charged and are propelled towards grounded collecting plates. After the particles are deposited in the plates, they lose their charge and the agglomerated dust is removed.

**Input Data:**
- Mass flowrate of component $i$ in the feed stream, $F_i$ [kg/h].
- Superficial gas velocity, $v$ [m/min].
- Number of ducts, $N_d$ [dimensionless].
- Number of electrical sections in the direction of gas flow, $N_s$ [dimensionless].
- Specification of either combination:
  - (i) plate height $H$ [m] and length $L$ [m], or
  - (ii) collection efficiency $\eta$ [dimensionless] and aspect ratio $R$ [dimensionless].
- Mean particle diameter in size range $i$, $d_i$ [$\mu$m].
- Mass fraction of particles in size range $i$, $m_i$ [dimensionless].
- Migration velocity of particles in size range $i$, $w_i$ [m/min].

**Output Data:**
- Mass flowrate of component $i$ in the clean stream, $C_i$ [kg/h].
- Mass flowrate of component $i$ in the dust stream, $D_i$ [kg/h].
- Plate separation, $D$ [m].
- Number of plates, $N_p$ [dimensionless].
- Total plate (collection) area, $A$ [m$^2$].

**Model:**
The total collection area $A$ is given by:
\[
A = 2 \, H \, L \, N_s \, N_d
\]

The aspect ratio is defined as:

\[
R = \frac{N_s \, L}{H}
\]

The plate separation \( D \) is calculated by:

\[
D = \frac{Q}{v \, N_d \, H}
\]

The number of plates \( N_p \) is given by:

\[
N_p = \frac{A}{A_p} + 1
\]

where \( A_p \) is the area per plate calculated by \( A_p = L \, H \).

The removal efficiency for each particle size \( \eta_i \), is estimated by the system using the Deutch equation:

\[
\eta_i = 100 \left[ 1 - \exp \left( -\frac{w_i \, A}{Q} \right) \right]
\]

The overall collection efficiency \( \eta \), is:

\[
\eta = \Sigma (m_i \, \eta_i)
\]

The mass flowrate of each component in the particulate stream is given by:
\[ P_i = F_i \eta_i \]

while the mass flowrate in the clean stream is:

\[ C_i = F_i (1 - \eta_i) \]

The solution algorithm for this model is given in Figure 7.18.

If overall collection efficiency, aspect ratio are specified

- Calculate total collection area
- Calculate plate height
- Calculate plate length
- Calculate collection area per plate

else

- Calculate aspect ratio
- Calculate collection area per plate
- Calculate total collection area
- Calculate collection efficiency per size range
- Calculate total collection efficiency

Calculate plate separation
Calculate number of plates
Do material balances
Estimate cost

**Figure 7.18** Solution Algorithm for the Electrostatic Precipitation Model
CHAPTER 8

DEVELOPMENT OF A DESIGN CASE IN ENVIROCAD

EnviroCAD is a fully graphical program. Its interactive interface is based on standard Macintosh features, such as icons, menus, and the use of the mouse. The interface is used for the development of a design case, i.e., the specification of flowsheet topology, entry of process data, and reporting of the calculated information. It also facilitates calculation of stream properties, process emissions and life cycle inventories, and auxiliary functionalities of the program (on-line help, save and load a design case, open reports, etc.).

The process of developing a design case in EnviroCAD, can be summarized in the following three steps:

• Create a flowsheet, i.e.,
  select the unit operations, and
  draw the process streams that connect the units.

• Initialize the flowsheet, i.e., provide input information about
  components,
  feed streams,
  unit operations, and
  cost data (optional).

• Analyze the flowsheet, i.e.,
  solve material and energy balances,
  generate stream report (optional), and
  perform economic analysis (optional).

A description of these steps, is presented in the following sections.
8.1 Creating a Flowsheet

A flowsheet is the basis of analysis in EnviroCAD. A process flowsheet consists of one or more unit operations and the streams that connect them.

8.1.1 Select Unit Operations

The selection of the unit operations is done through the Unit_Ops menu (Figure 8.1). Each unit operation has a unique icon. Once on the screen, a unit can be moved around, flipped horizontally or erased, using items from the palette appearing on the left side of the screen. Names of units are assigned directly by the program, but they are editable.

![Figure 8.1 The Unit Operations Menu](image-url)
8.1.2 Draw Streams

Each unit operation has inlet and outlet ports for connection of streams. The drawing of the streams is done by using the \( \Rightarrow \) palette item. There are feed, intermediate and product streams. Each category, has a different representation on the flowsheet (see Figure 8.2). Each stream is assigned a unique number automatically, which can be changed, if needed.

![Stream Representation in EnviroCAD](image)

**Figure 8.2** Stream Representation in EnviroCAD

8.2 Initializing a Flowsheet

Flowsheet initialization is done using the first three items of the Tasks menu (Figure 8.3). Initialization of components, feed streams, and unit operations is required, while the initialization of the cost data is optional. Initialization of the feed streams and unit operations can be also done by clicking on the respective stream and unit operation icons using the lens cursor (see palette).
8.2.1 **Initialize Components**

By selecting this task, a series of dialog windows appear that enable the specification of the chemical components that are used to represent material flows in the flowsheet. The physical and environmental properties of the chemical components are specified through the same dialog windows. Figure 8.4 presents the first of these initialization windows. Default values are provided for some frequently used components. Additional components can be easily added.

8.2.2 **Initialize Feed Streams**

For each feed stream, the temperature, pressure and mass flowrate for each component must be specified. By selecting this item from the **Tasks** menu, a dialog window similar to the one shown in Figure 8.5, is displayed for each feed stream.
Figure 8.4 First Dialog Window for Initialization of Components

Figure 8.5 Dialog Window for Initialization of Feed Streams
8.2.3 Initialize Unit Operations

The selection of this task displays a series of dialog windows specific to each unit operation in the flowsheet. These windows contain input and output design information for each unit. For instance, Figure 8.6 presents the first dialog window used for the initialization of a clarifier. Only the input variables are editable. For all the input variables, there are default values that can be used for a first run or until better values become available. These default values are derived from generally accepted average values that appear in the literature.

![CR-101 Design Data](image)

**Figure 8.6** First Dialog Window of the Clarifier Model

8.2.4 Initialize Cost Data

The initialization of the data needed for the economic evaluation of the flowsheet are provided during this step, using the *Get Cost Data* item of the *Tasks* menu.
8.3 Analyzing a Flowsheet

Flowsheet analysis includes the calculation of the material and energy balances, the sizing and costing of the equipment, and the generation of various reports. The options for flowsheet analysis are found under the Tasks menu (Figure 8.3).

8.3.1 Solve M&E Balances

This task carries out the material and energy balances, calculates the lumped stream environmental properties, and estimates the size and purchase cost of each unit operation. The results of the simulation can be viewed immediately by clicking on streams and unit operations using the lens cursor.

8.3.2 Generate Stream Report

This task is optional. It generates an output text file which presents a detailed stream report. For each stream, the name, source and destination unit operations, flowrate of each component and environmental stream properties are reported. A sample stream report is presented in Chapter 9.

8.3.3 Perform Economic Analysis

This task is also optional. It creates an output text file which presents the economic analysis for the entire flowsheet. The estimated purchase cost for each unit operation is also found here. A sample economic analysis report is presented in Chapter 9.

8.4 Process Emissions

The module for calculating process emissions is under the Design menu (Figure 8.7). The initialization steps for the calculation of process emissions include:
• Specification of the chemical components considered for emissions calculations (Figure 8.8a).

• Specification of the unit operations considered in the emissions calculations (Figure 8.8b).

• Specification of the emission generating operating task(s) for each unit operation selected, using the dialog window shown in Figure 8.8c.

• Initialization of the operating task(s) through appropriate dialog windows. For example, the dialog window for the initialization of charging is shown in Figure 8.8d.

Once the above data have been specified, the process emissions can be calculated and reported in the emissions report.

Figure 8.7 The Emissions Menu
**Figure 8.8a** Specification of Components Considered in Emissions Calculations

**Figure 8.8b** Selection of Unit Operations Considered for Emissions Calculations
Specify Emission Tasks for Unit: R-101

- Charging
- Evacuation
- Purging
- Heating
- Gas Evolution
- Vacuum
- Drying

Figure 8.8c Dialog Window for the Selection of Tasks

R-101 Design Data

Charging Data for: R-101

Liquid Pumping Rate (SCFM): 30.00
Exit Temperature (°C): 30.0

Figure 8.8d Dialog Window for the Initialization of the Charging Task
8.5 Life Cycle Inventory Analysis

The module for carrying out the life cycle inventory analysis, is also found under the Design menu (Figure 8.9). The initialization steps for the calculation of environmental load inventories are:

- Identification of product streams (Figure 8.10).
- Initialization of the mass eco-vectors for feed streams (see Figure 6.4).
- Initialization of the mass eco-vectors for waste streams.
- Initialization of energy eco-vectors (see Figure 6.5).

Once the initialization step is complete, the inventory analysis can be performed and reported.

Figure 8.9 The Life Cycle Inventory Analysis Menu
### Figure 8.10 Characterization of Output Streams

<table>
<thead>
<tr>
<th>Stream Name</th>
<th>Product/Waste (P/W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$102</td>
<td>Product</td>
</tr>
<tr>
<td>$104</td>
<td>Waste</td>
</tr>
<tr>
<td>$105</td>
<td>Waste</td>
</tr>
</tbody>
</table>

**Legend:**
- **Product**
- **Waste**
EXAMPLE: DESIGN OF A POLLUTION CONTROL FACILITY

This chapter demonstrates the application of the developed computer tools for the design of an industrial wastewater treatment plant. Based on plant influent, the treatment facility is first synthesized by WasteMin Expert and then analyzed using EnviroCAD.

9.1 Synthesis of Wastewater Treatment Facility

Table 9.1 shows the composition of the influent stream. It corresponds to an average treatment plant with a throughput of 5 million gallons per day. All the influent organic compounds are regulated under the Resource Conservation and Recovery Act (RCRA).

<table>
<thead>
<tr>
<th>Component</th>
<th>Flowrate (kg/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water</td>
<td>787,000</td>
</tr>
<tr>
<td>Ammonia</td>
<td>10</td>
</tr>
<tr>
<td>Napthalene</td>
<td>70</td>
</tr>
<tr>
<td>Acetone</td>
<td>25</td>
</tr>
<tr>
<td>Methylethylketone</td>
<td>45</td>
</tr>
<tr>
<td>Glucose</td>
<td>140</td>
</tr>
<tr>
<td>Chlorobenzene</td>
<td>45</td>
</tr>
<tr>
<td>Tetrahydrofuran</td>
<td>20</td>
</tr>
<tr>
<td>Trichloroethylene</td>
<td>35</td>
</tr>
<tr>
<td>1,1,2 Trichloroethane</td>
<td>18</td>
</tr>
<tr>
<td>Ethylbenzene</td>
<td>20</td>
</tr>
<tr>
<td>Phosphoric Acid</td>
<td>50</td>
</tr>
<tr>
<td>Toluene</td>
<td>25</td>
</tr>
<tr>
<td>Arsenic</td>
<td>2</td>
</tr>
</tbody>
</table>
Based on this stream, WasteMin Expert interactively synthesizes the wastewater treatment facility. The questions asked by the system are shown in Table 9.2, while the user input and system response to each question are shown in Table 9.3. User inputs are also discussed briefly in Table 9.3. System responses are based on the decision diagrams presented in Chapter 4. The resulting conceptual design is shown schematically in Figure 9.1.

Table 9.2 Questions asked by WasteMin Expert

<table>
<thead>
<tr>
<th>Pre-Treatment</th>
<th>1. Do you anticipate any extreme variations in the flowrate or composition of the wastestream?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. What is the pH of the wastestream?</td>
</tr>
<tr>
<td>Primary</td>
<td>3. Would you like to consider coagulation/flocculation for enhanced suspended solids removal?</td>
</tr>
<tr>
<td>Treatment</td>
<td>4. What is the desired clarification area (m²)?</td>
</tr>
<tr>
<td>Secondary</td>
<td>5. Would you like to consider suspended growth biological processes for the secondary treatment (as opposed to attached growth processes)?</td>
</tr>
<tr>
<td>Treatment</td>
<td>6. Is there land available for an aerated lagoon?</td>
</tr>
<tr>
<td>Tertiary</td>
<td>7. Would you like distinct aeration and sedimentation steps for the biological treatment?</td>
</tr>
<tr>
<td>Treatment</td>
<td>8. Would you like to consider tertiary (advanced) treatment?</td>
</tr>
<tr>
<td>Sludge</td>
<td>9. Would you like to consider sludge thickening?</td>
</tr>
<tr>
<td>Treatment</td>
<td>10. Would you like to consider combined treatment of sludges from all wastewater treatment steps?</td>
</tr>
<tr>
<td></td>
<td>11. Sludge conditioning is used in advance of mechanical dewatering systems. Would you like to consider sludge conditioning?</td>
</tr>
<tr>
<td></td>
<td>12. Solid bowl centrifugation will be used for sludge dewatering. Would you rather consider filtration for dewatering?</td>
</tr>
<tr>
<td></td>
<td>13. Would you like to use continuous filtration (vs. batch)?</td>
</tr>
<tr>
<td>Sludge</td>
<td>14. Would you like filter operation under vacuum?</td>
</tr>
<tr>
<td>Disposal</td>
<td>15. Would you like sludge disposal by combustion (also recommended for sludges with a high content of hazardous materials)?</td>
</tr>
</tbody>
</table>

### Table 9.3 User Input and WasteMin Expert Responses

<table>
<thead>
<tr>
<th>Q #</th>
<th>User Input - Explanation</th>
<th>WasteMin Expert Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes. (This particular reasoning is specific to plant in question)</td>
<td>Equalization.</td>
</tr>
<tr>
<td>2</td>
<td>4.3</td>
<td>Neutralization of the acidic wastestream using one of the following agents: Ca(OH)$_2$, NaOH, CaCO$_3$.</td>
</tr>
<tr>
<td>3</td>
<td>Yes. (Suspended solids are a major concern in this plant)</td>
<td>40% of suspended solids are removed by primary sedimentation.</td>
</tr>
<tr>
<td>4</td>
<td>50 (An average clarification area)</td>
<td>The activated sludge process is used for biological treatment. A clarifier is used for the sedimentation of the biological solids.</td>
</tr>
<tr>
<td>5</td>
<td>Yes. (Suspended growth processes are most commonly used)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>No. (Area is limited in this plant)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Yes. (Distinct steps have better treatment results)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>No. (Secondary treatment may be adequate)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Yes. (Need to reduce the sludge volume)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Yes. (Sludge from primary treatment is negligible in this particular case)</td>
<td>The sludge is thickened by a gravity thickener. Anaerobic digestion is used for sludge stabilization.</td>
</tr>
<tr>
<td>11</td>
<td>Yes. (Mechanical dewatering will be used)</td>
<td>Chemical conditioning of the sludge is achieved.</td>
</tr>
<tr>
<td>12</td>
<td>Yes. (Filtration is preferable in this case)</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Yes. (Sludge production is large, which favors the continuous process)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>No. (Vacuum has a higher operational cost)</td>
<td>A belt filter is used for sludge dewatering.</td>
</tr>
<tr>
<td>15</td>
<td>No. (Incineration is costly)</td>
<td>Landfilling is used for sludge disposal.</td>
</tr>
</tbody>
</table>

![Figure 9.1 Conceptual Flowsheet Generated by WasteMin Expert](image-url)
9.2 Analysis of Wastewater Treatment Facility

The output of WasteMin Expert is used to guide the development of a treatment flowsheet in EnviroCAD. Based on the list of unit operations generated by the expert system, a detailed flowsheet is developed by the user, which contains loops, stream splits and detailed design information. The next paragraphs describe the resulting flowsheet, which is shown in Figure 9.2.

Pre-treatment starts with the equalization basin (V-101) which has a volume of 2,000 m$^3$. The pH of the acidic stream (S102) is adjusted to around 7 in a neutralizer (R-101) with lime (S104). Continuous operation is assumed for the neutralizer with a residence time of 1 hour. During primary treatment, a primary clarifier (CR-103) is used for the removal of the suspended solids. The resulting concentration of solids in the clarifier underflow (S107) is 2% w/w.

The secondary treatment starts with an aeration basin (R-102) with a hydraulic residence time of 6 hours, for the biological oxidation of organic materials. It also includes a secondary clarifier (CR-101) for the removal of sludge and carried-over solids. The solids concentration in the sludge effluent (S110) from the secondary clarifier is 0.8% w/w. A fraction of the sludge (30%) is recycled (S112) to maintain a biomass concentration in the aeration basin of 3,500 mg/liter.

The excess sludge (S113) from the secondary clarifier and the underflow from the primary clarifier (S107) are sent to the sludge treatment section. The sludge is thickened in a thickener (CR-102) to a solids concentration of 4% w/w, digested in an anaerobic digester (R-103) to a solids concentration of 12% w/w, and finally dewatered in a belt filter press (BFP-101) to a solids concentration of 20% w/w. Landfilling of the sludge follows.
Figure 9.2 Wastewater Treatment Facility
Once the treatment flowsheet is specified, the key question is the fate of each of the chemicals present in the feed stream. Water soluble and easily biodegradable substances will be oxidized in the aeration basin by bacteria and other microorganisms and converted into CO$_2$ and H$_2$O. A fraction of recalcitrant and non-biodegradable compounds with low water solubility will be sorbed on biomass and follow its path through the treatment plant. Finally, a fraction of volatile compounds will be stripped off and end up in the atmosphere, resulting in secondary emissions.

The component specific removal fractions in the aeration basin, used for the simulation of this plant in EnviroCAD, are shown in Table 9.4. These fractions were taken from the experimental results by EPA (1989).

<table>
<thead>
<tr>
<th>Component</th>
<th>% Biodegradation</th>
<th>% Stripping</th>
<th>% Sorption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acetone</td>
<td>94.1</td>
<td>1.8</td>
<td>0.6</td>
</tr>
<tr>
<td>Methylethylketone</td>
<td>94.3</td>
<td>0.8</td>
<td>0.6</td>
</tr>
<tr>
<td>Tetrahydrofuran</td>
<td>77.9</td>
<td>10.2</td>
<td>1.2</td>
</tr>
<tr>
<td>Chlorobenzene</td>
<td>83.6</td>
<td>12.6</td>
<td>2.8</td>
</tr>
<tr>
<td>Trichloroethylene</td>
<td>45.5</td>
<td>52.4</td>
<td>1.8</td>
</tr>
<tr>
<td>1,1,2-Trichloroethane</td>
<td>31.7</td>
<td>37.0</td>
<td>2.1</td>
</tr>
<tr>
<td>Ethylbenzene</td>
<td>77.2</td>
<td>15.9</td>
<td>5.1</td>
</tr>
<tr>
<td>Toluene</td>
<td>72.0</td>
<td>25.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Phenol</td>
<td>97.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Naphtalene</td>
<td>94.0</td>
<td>0.0</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Table 9.5 shows the overall performance of the plant based on the environmental stream properties calculated by EnviroCAD. This plant removes around 85% of BOD$_5$ and 65% of TSS. Based on the minimum standards for secondary treatment set by EPA, the average removal efficiency for both BOD$_5$ and TSS shall not be less than 85%.
Therefore, further treatment must be considered to enhance plant performance, especially in terms of TSS removal.

Table 9.5 Overall Performance Based on Environmental Stream Properties (kg/day)

<table>
<thead>
<tr>
<th>Environmental Stream Property</th>
<th>Influent (S101)</th>
<th>Effluent Water (S126)</th>
<th>Effluent Sludge (S122)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Organic Carbon</td>
<td>7,542</td>
<td>783</td>
<td>476</td>
</tr>
<tr>
<td>Chemical Oxygen Demand</td>
<td>24,585</td>
<td>2,897</td>
<td>1,773</td>
</tr>
<tr>
<td>Biochemical Oxygen Demand</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ultimate</td>
<td>15,540</td>
<td>2,001</td>
<td>1,630</td>
</tr>
<tr>
<td>5-day</td>
<td>8,140</td>
<td>1,217</td>
<td>1,108</td>
</tr>
<tr>
<td>Total Kjeldahl Nitrogen</td>
<td>471</td>
<td>106</td>
<td>111</td>
</tr>
<tr>
<td>Total Phosphorus</td>
<td>408</td>
<td>59</td>
<td>289</td>
</tr>
<tr>
<td>Total Suspended Solids</td>
<td>2,448</td>
<td>844</td>
<td>2,191</td>
</tr>
</tbody>
</table>

In addition, this plant performs rather poorly as far as emissions of VOCs and heavy metals release are concerned. Table 9.6 presents the composition of the gas stream exiting the aeration basin (S106). From all the VOCs present in this table, only acetone and tetrahydrofuran are not classified as Hazardous Air Pollutants (HAPs) under Title III of the 1990 Clean Air Act Amendments (CAAA) (Kao, 1994). A total of approximately 40 kg/h or 960 kg/day of HAPs are emitted from this single unit operation. Additional secondary emissions from the equalizer, neutralizer and clarifiers are ignored, due to lack of information. According to the CAAA, a plant is considered a major source of HAPs if it has the potential to emit 10 tons/year or more of any single HAP or 25 tons/year or more of any combination of HAPs. Clearly, HAPs emissions from this plant exceed the 25 tons/year limit. In order to comply with this limit,
application of Reasonably Available Control Technology (RACT) would be required for this wastewater treatment plant, to reduce the amount of VOCs emissions (McInnes, 1993). Such technologies include wet scrubbing (packed or mist towers), adsorption (based on activated carbon or zeolites), condensation, biofiltration, membrane filtration, and thermal or catalytic oxidation.

Table 9.6 Aeration Basin Gas Outlet (stream S106)

<table>
<thead>
<tr>
<th>Component</th>
<th>Flowrate (kg/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carbon Dioxide</td>
<td>90.97</td>
</tr>
<tr>
<td>Ammonia</td>
<td>8.95</td>
</tr>
<tr>
<td>Acetone</td>
<td>0.45</td>
</tr>
<tr>
<td>MethylEthylKetone</td>
<td>0.36</td>
</tr>
<tr>
<td>Chlorobenzene</td>
<td>5.63</td>
</tr>
<tr>
<td>Tetrahydrofuran</td>
<td>2.03</td>
</tr>
<tr>
<td>Trichloroethylene</td>
<td>18.22</td>
</tr>
<tr>
<td>1,1,2 Trichloroethane</td>
<td>6.64</td>
</tr>
<tr>
<td>Ethylbenzene</td>
<td>3.16</td>
</tr>
<tr>
<td>Toluene</td>
<td>6.21</td>
</tr>
<tr>
<td>HAPs Total</td>
<td>40.22</td>
</tr>
</tbody>
</table>

Table 9.7 presents the effluent from the overflow of the secondary clarifier (S126). The arsenic concentration of 2.2 ppm in the effluent stream is above the current federal limit, set by the Safe Drinking Water Act, of 50 µg/liter. Heavy metals, if present at high concentrations, can be removed efficiently by precipitation. For dilute streams on the other hand, ion exchange is usually more effective and economical. Strong base
anion exchangers have been found by Vance (1995), to give a 99+ percent soluble arsenic removal from industrial wastewater.

**Table 9.7 Secondary Clarifier Overflow Stream (S126)**

<table>
<thead>
<tr>
<th>Component</th>
<th>Flowrate (kg/h)</th>
<th>Concentration (ppm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water</td>
<td>755,841.4</td>
<td>999,800.0</td>
</tr>
<tr>
<td>Biomass</td>
<td>31.77</td>
<td>42.03</td>
</tr>
<tr>
<td>Glucose</td>
<td>13.27</td>
<td>17.56</td>
</tr>
<tr>
<td>Ammonia</td>
<td>0.95</td>
<td>1.25</td>
</tr>
<tr>
<td>Napthalene</td>
<td>3.98</td>
<td>5.27</td>
</tr>
<tr>
<td>Acetone</td>
<td>0.97</td>
<td>1.28</td>
</tr>
<tr>
<td>MethylEthylKetone</td>
<td>2.09</td>
<td>2.76</td>
</tr>
<tr>
<td>Chlorobenzene</td>
<td>1.62</td>
<td>2.15</td>
</tr>
<tr>
<td>Tetrahydrofuran</td>
<td>2.26</td>
<td>2.99</td>
</tr>
<tr>
<td>Trichloroethylene</td>
<td>0.70</td>
<td>0.92</td>
</tr>
<tr>
<td>1,1,2 Trichloroethane</td>
<td>5.36</td>
<td>7.09</td>
</tr>
<tr>
<td>Ethylbenzene</td>
<td>1.31</td>
<td>1.73</td>
</tr>
<tr>
<td>Toluene</td>
<td>0.71</td>
<td>0.94</td>
</tr>
<tr>
<td>Arsenic</td>
<td>1.65</td>
<td>2.2</td>
</tr>
<tr>
<td>Lime</td>
<td>10.88</td>
<td>14.39</td>
</tr>
<tr>
<td>Phosphoric Acid</td>
<td>4.8</td>
<td>6.35</td>
</tr>
<tr>
<td>Calcium Phosphate</td>
<td>1.94</td>
<td>2.57</td>
</tr>
</tbody>
</table>

These observations form the basis of a retrofit process design. In the next section, process modifications are evaluated for (i) enhanced removal of suspended solids, (ii)
control of VOCs emissions from the aeration basin, and (ii) removal of heavy metals from the water effluent stream.

9.3 Process Modifications

Figure 9.3 shows the modified flowsheet. A granular media filter (GMF-101) has been added for additional removal of suspended solids, followed by an ion exchanger (C-102) for the removal of arsenic. An activated carbon adsorption unit (C-101) is considered for VOCs emissions control. It is assumed that the aeration basin is covered with a roof to collect the exiting gases. The carbon column is designed to remove at least 98% of the VOCs. Table 9.8 presents the performance comparison between the original and the modified flowsheet in terms of the treatment objectives. The modified flowsheet has a 95.6% BOD$_5$ removal, a 99.6% TSS removal, and meets the standards for arsenic (effluent concentration of 40 µg/liter) and HAPs (less than 25 tons/year emissions).

Table 9.8 Performance Comparison Between the Original and the Modified Flowsheet

<table>
<thead>
<tr>
<th>Treatment Objective</th>
<th>Influent ($S101$)</th>
<th>Original Flowsheet</th>
<th>Modifed Flowsheet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Effluent ($S126$)</td>
<td>% Removal</td>
<td>Effluent ($S134$)</td>
</tr>
<tr>
<td>BOD$_5$ (kg/day)</td>
<td>8,140</td>
<td>1,217</td>
<td>357</td>
</tr>
<tr>
<td>TSS (kg/day)</td>
<td>2,448</td>
<td>844</td>
<td>8.4</td>
</tr>
<tr>
<td>Arsenic (µg/l)</td>
<td>2,540</td>
<td>2,190</td>
<td>40</td>
</tr>
</tbody>
</table>

Aeration Basin Gas Exit ($S106$) Aeration Basin Gas Exit ($S127$)

HAPs (kg/day) 965.3 19.3
Figure 9.3 Modified Wastewater Treatment Facility
EnviroCAD is also used to carry out an economic evaluation of the two alternative flowsheets. Key assumptions made for the economic evaluation include:

- An increase of 20% in the capital cost of the aeration basin, to account for the addition of the roof required to collect all exiting gases and direct them to the adsorption unit.
- For the activated carbon adsorption unit, a carbon cost of $3/kg and a replacement frequency of once every 40,000 hours.
- An average disposal cost of $0.75/kg of VOCs for the mixture of organic solvents recovered by the activated carbon adsorption unit.
- For the ion exchanger, a resin unit cost of $10/liter and a replacement frequency of once every 4 years.
- For the disposal of the heavy metals recovered by the ion exchanger, a cost of $1/kg of solution (heavy metals concentration was 0.95% w/w).
- A increase of 20% in the total purchase equipment cost, to account for the cost of the unlisted equipment. The cost of pumps, some process tanks not shown on the flowsheet, etc. are included under this category.

Table 9.9 shows the cost of each case, as calculated by the economic evaluation module of EnviroCAD. The modified flowsheet has higher capital and operating costs.

<table>
<thead>
<tr>
<th>Flowsheet</th>
<th>Direct Fixed Capital Cost ($M)</th>
<th>Operating Cost ($/kg of BOD5 removed)</th>
<th>Operating Cost ($/kg/year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>9.9</td>
<td>1.35</td>
<td>3.3</td>
</tr>
<tr>
<td>Modified</td>
<td>15.2</td>
<td>2.08</td>
<td>5.9</td>
</tr>
</tbody>
</table>
Tables 9.10a, b, c are excerpts from the economic evaluation report of the modified flowsheet. Table 9.10a shows a list of the plant equipment with some descriptive information and the purchase cost of each piece. The total equipment purchase cost is around $3.6 million.

Table 9.10b shows a summary of the fixed capital investment which for the case of the modified flowsheet is $15.2 million. Due to the lack of reliable data, the cost multipliers used for the calculation of the fixed capital investment were adapted and modified from projects for manufacturing processes (Valle-Riestra, 1983).

A summary of the annual operating cost is presented in Table 9.10c. Clearly, the direct fixed capital (DFC) dependent cost is the most important item followed by the labor dependent cost. A $20/hour rate is assumed for operating labor. The cost of raw materials includes the cost of Ca(OH)$_2$ (at $0.2/kg) used in the neutralizer. The waste disposal cost includes the cost of carbon disposal from the adsorption column and the stabilization of the heavy metals removed by the ion exchange. All the multipliers used to estimate various items of the fixed capital and operating costs are editable.

This example illustrates how EnviroCAD can be used to:

- quantify environmental loads and track the fate of regulated chemicals, and
- evaluate process modifications and extensions necessitated by new regulations.

It also demonstrates clearly, that the pollution control cost of a manufacturing facility, especially in cases where volatile organics and heavy metals are present in the feed stream, can be quite substantial, contributing a major fraction of the overall operating cost. Instead of trying to remove such hazardous chemicals from dilute wastewater streams, emphasis should be placed on their recovery and recycle inside the manufacturing battery limits. Also, this cost is an incentive to design new as well as existing facilities so that they produce less waste. In other words, pollution prevention should always be the primary objective.
Table 9.10a Major Equipment Specification and FOB Cost (1995 Prices)

<table>
<thead>
<tr>
<th>Quantity/Stand-by</th>
<th>Description</th>
<th>Unit Cost ( $ )</th>
<th>Cost ( $ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/0 V-101</td>
<td>Equalizer</td>
<td>614000</td>
<td>614000</td>
</tr>
<tr>
<td></td>
<td>Volume = 2000.00 m³</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2/0 R-101</td>
<td>Neutralizer</td>
<td>197000</td>
<td>394000</td>
</tr>
<tr>
<td></td>
<td>CS, Volume = 471.71 m³</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/0 CR-103</td>
<td>Primary Clarifier</td>
<td>119000</td>
<td>119000</td>
</tr>
<tr>
<td></td>
<td>Vol = 1603.81 m³, Depth = 8.40 m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/0 CR-101</td>
<td>Secondary Clarifier</td>
<td>119000</td>
<td>119000</td>
</tr>
<tr>
<td></td>
<td>Vol = 3972.80 m³, Depth = 5.00 m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/0 GMF-101</td>
<td>Granular Media Filter</td>
<td>110000</td>
<td>660000</td>
</tr>
<tr>
<td></td>
<td>Diam = 2.99 m, Flux = 300.00 L/m²/min</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/0 C-102</td>
<td>Ion Exchange Unit</td>
<td>295000</td>
<td>295000</td>
</tr>
<tr>
<td></td>
<td>L = 4.00 m, D = 0.98 m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/0 CR-102</td>
<td>Thickener</td>
<td>77000</td>
<td>77000</td>
</tr>
<tr>
<td></td>
<td>Area = 45.35 m²</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/0 R-103</td>
<td>Anaerobic Digestion</td>
<td>209000</td>
<td>209000</td>
</tr>
<tr>
<td></td>
<td>Volume = 3046.92 m³</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Loading = 0.10 kg BOD5/day</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/0 BFP-101</td>
<td>Belt Filter</td>
<td>107000</td>
<td>107000</td>
</tr>
<tr>
<td></td>
<td>Width = 2.00 m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/0 R-104</td>
<td>Aeration Basin</td>
<td>211000</td>
<td>211000</td>
</tr>
<tr>
<td></td>
<td>Volume = 6459.94 m³</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Loading = 19987.91 kg BOD5/day</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2/0 C-101</td>
<td>Adsorption Column</td>
<td>22000</td>
<td>44000</td>
</tr>
<tr>
<td></td>
<td>Diam = 0.51 m, Length = 1.53 m</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cost of Unlisted Equipment 20.0 % of Total

-------------------------------
TOTAL EQUIPMENT PURCHASE COST 3561000
-------------------------------
Table 9.10b Fixed Capital Estimate Summary (1995 Prices)

A. TOTAL PLANT DIRECT COST (TPDC) (physical cost)

<table>
<thead>
<tr>
<th>Item</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Equipment Purchase Cost (PC)</td>
<td>$3561000</td>
</tr>
<tr>
<td>2. Installation (0.24 X PC)</td>
<td>866000</td>
</tr>
<tr>
<td>3. Process Piping (0.35 X PC)</td>
<td>1246000</td>
</tr>
<tr>
<td>4. Instrumentation (0.15 X PC)</td>
<td>534000</td>
</tr>
<tr>
<td>5. Insulation (0.03 X PC)</td>
<td>107000</td>
</tr>
<tr>
<td>6. Electrical (0.10 X PC)</td>
<td>356000</td>
</tr>
<tr>
<td>7. Buildings (0.15 X PC)</td>
<td>534000</td>
</tr>
<tr>
<td>8. Yard Improvement (0.15 X PC)</td>
<td>534000</td>
</tr>
<tr>
<td>9. Auxiliary Facilities (0.15 X PC)</td>
<td>534000</td>
</tr>
<tr>
<td><strong>TPDC</strong></td>
<td><strong>8272000</strong></td>
</tr>
</tbody>
</table>

B. TOTAL PLANT INDIRECT COST (TPIC)

<table>
<thead>
<tr>
<th>Item</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>10. Engineering (0.25 X TPDC)</td>
<td>2068000</td>
</tr>
<tr>
<td>11. Construction (0.35 X TPDC)</td>
<td>2895000</td>
</tr>
<tr>
<td><strong>TPIC</strong></td>
<td><strong>4963000</strong></td>
</tr>
</tbody>
</table>

C. TOTAL PLANT COST (TPDC + TPIC)

<table>
<thead>
<tr>
<th>Item</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>12. Contractor's fee  (0.05 X TPC)</td>
<td>662000</td>
</tr>
<tr>
<td>13. Contingency       (0.10 X TPC)</td>
<td>1323000</td>
</tr>
<tr>
<td><strong>∑(12+13)</strong></td>
<td><strong>1985000</strong></td>
</tr>
</tbody>
</table>

D. DIRECT FIXED CAPITAL (DFC) TPC + 12 + 13

<table>
<thead>
<tr>
<th>Item</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DFC</strong></td>
<td><strong>1522000</strong></td>
</tr>
</tbody>
</table>
### Table 9.10c Annual Operating Cost (1995 Prices)

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Cost (1995)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. DFC-DEPENDENT ITEMS (DFC = $ 15220000)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Depreciation</td>
<td>$ 1446000</td>
<td></td>
</tr>
<tr>
<td>Maintenance Material</td>
<td>(0.03 X DFC)</td>
<td>457000</td>
</tr>
<tr>
<td>Insurance</td>
<td>(0.01 X DFC)</td>
<td>152000</td>
</tr>
<tr>
<td>Local Taxes</td>
<td>(0.02 X DFC)</td>
<td>304000</td>
</tr>
<tr>
<td>Factory Expense</td>
<td>(0.05 X DFC)</td>
<td>761000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3120000</td>
</tr>
<tr>
<td><strong>2. LABOR-DEPENDENT ITEMS</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operating labor</td>
<td>(8760 h X 20.0 $/h)</td>
<td>175000</td>
</tr>
<tr>
<td>Maintenance labor</td>
<td>(0.03 X DFC)</td>
<td>457000</td>
</tr>
<tr>
<td>Fringe Benefits</td>
<td>(0.40 X (a+b))</td>
<td>253000</td>
</tr>
<tr>
<td>Supervision</td>
<td>(0.20 X (a+b))</td>
<td>126000</td>
</tr>
<tr>
<td>Operating supplies</td>
<td>(0.10 X a)</td>
<td>17000</td>
</tr>
<tr>
<td>Laboratory</td>
<td>(0.15 X a)</td>
<td>26000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1054000</td>
</tr>
<tr>
<td><strong>3. ADMINISTRATION AND OVERHEAD EXPENSE</strong></td>
<td>(0.6 X (a+b+c))</td>
<td>455000</td>
</tr>
<tr>
<td><strong>4. RAW MATERIALS</strong></td>
<td></td>
<td>99000</td>
</tr>
<tr>
<td><strong>5. UTILITIES</strong></td>
<td></td>
<td>119000</td>
</tr>
<tr>
<td><strong>6. WASTE DISPOSAL</strong></td>
<td></td>
<td>1052000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL ANNUAL OPERATING COST</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Including Depreciation</td>
<td></td>
<td>5899000</td>
</tr>
<tr>
<td>Excluding Depreciation</td>
<td></td>
<td>4453000</td>
</tr>
</tbody>
</table>
CHAPTER 10

CONTRIBUTIONS AND RECOMMENDATIONS

10.1 Research Results

Problems related to process design with environmental considerations have been studied in this dissertation. The contributions of this research include:

- A new methodology for the minimization of the adverse environmental impact of chemical processes. This hierarchical methodology starts with the definition and analysis of a process flowsheet; quantification of the process environmental loads using proper entities follows. If the calculated environmental loads do not satisfy the environmental constraints, waste minimization is initially investigated via pollution prevention. Process modifications, aiming at pollution prevention, are identified, prioritized, and evaluated. If pollution prevention is not feasible, pollution control is the next level in the waste minimization hierarchy of the methodology. A pollution control flowsheet is synthesized, analyzed, and evaluated based on the environmental loads. Computer tools that facilitate the design of pollution control flowsheets have been developed.

- WasteMin Expert, a knowledge-based system which provides an organized framework for assisting the designer in the synthesis of pollution control flowsheets. Its knowledge base consists of heuristics compiled from several literature sources. It is an interactive rule-based expert system, which follows the task decomposition paradigm to accomplish the flowsheet synthesis task. The output of WasteMin Expert provides an ordered list of key unit operations for pollution control. This list serves as a good starting point for the creation of detailed designs using a process simulator.
EnviroCAD, an environmental simulator which includes a variety of unit operation models, is used for the analysis and evaluation of integrated pollution control processes. EnviroCAD follows the sequential modular approach and its models are based on standard design variables incorporated into appropriate algorithms. EnviroCAD accounts for environmental loads on the basis of individual pollutants as well as aggregate environmental stream properties. This feature of the program facilitates process retrofitting efforts to achieve the environmental objectives.

This thesis represents a first step in the research work required to define and satisfy the needs for process design with environmental constraints. Some areas in which further work may be useful are highlighted in the next section.

10.2 Recommendations for Future Work

Waste minimization efforts are facilitated by the quantification of pollutant release to the environment and ultimately quantification of the environmental impact of those releases. Although there are methodologies, indicators and properties that have been used to address this issue, the ideal case would be the existence of a single quantitative measure that represents the overall environmental impact of an activity, process, stream, etc. Future research in the field should focus towards this direction.

With respect to the computer tools, since both of them are research prototypes, a number of improvements and extensions are needed to make them more complete, robust, and effective. However, one should keep in mind that there is always a trade-off between system performance and simplicity. One of the major concerns in the development of any computer-aided system should be to find the optimum between performance and ease-of-use.
For WasteMin Expert, the following are a few suggestions for improvement:

- A more complete set of rules for the tasks currently addressed by the system.
- An extension of its knowledge base towards air pollution control issues.
- An advisory facility to help the user make decisions within the interactive framework of WasteMin Expert.
- An interface to EnviroCAD, so that the flowsheets synthesized by the expert system can be transferred automatically to the simulator.

Changes and additions to the current implementation of EnviroCAD could include:

- Use of kinetic models for the prediction of the fate of chemicals in the integrated pollution control facilities. These models should be based on fundamental principles for estimating the extent of biodegradation, sorption, and stripping for a compound that enters the treatment plant. High priority should be placed on the development of more rigorous biological treatment unit operation models. However, this improvement is closely related to the research efforts for the development of fundamental models, and the availability of kinetic data, which is now the limiting step towards this direction.

- Linkage of the simulator to a component database, to facilitate component initialization which currently is a tedious and time consuming task, due to the large number of properties required for the purposes of environmental simulation.

- In the current implementation of the life cycle inventory analysis module, if there are more than one product streams, the distribution of the environmental loads among them is based on their relative mass flowrates. However, this may not be always an objective way for distributing the loads. For example in the waste intensive pharmaceutical industry where low volume main products are the case, the distribution factor for the main product has to be much higher than its production
volume indicates. In the future, the user should have the option to assign the environmental load distribution factors among the product streams.

- Incorporation of a Total Cost Assessment module in the economic evaluation component of EnviroCAD. The limited availability of information about hidden environmental costs is currently the impediment to its implementation. This module should also be able to handle uncertainty.

- Along the same lines, more research is needed to acquire reliable waste management cost data in order to verify the cost multipliers used for economic evaluation in EnviroCAD.
APPENDIX I

UNIT OPERATION DATA STRUCTURES IN ENVIROCAD
struct Equalizer {
    short NumOfIntervals; /* Number of sampling intervals */
    float InterDuration; /* Time duration of each interval, min */
    float InfluentConc[25]; /* Avg Influent Concentration in each period */
    float EffluentConc[25]; /* Avg Effluent Concentration in each period */
    float AvgFlow[25]; /* Avg inflow during each period, m3/min */
    float AvgOutFlow; /* Equalized Output Flowrate, m3/min */
    float BasinVolume; /* Volume of equalization basin, m3 */
    float InfluentPF; /* Influent Peaking Factor */
    float EffluentPF; /* Effluent Peaking Factor */
    Boolean UserSpecVol; /* TRUE if Volume is User Specified */
};

typedef struct Equalizer EQUALIZER;
typedef EQUALIZER *EqualizerPtr;

struct Clarifier {
    Boolean OverflowSpec; /* TRUE if Overflow is User Specified */
    Boolean DepthSpec; /* TRUE if depth is specified */
    Boolean Circular; /* TRUE if circular design is used */
    float LiqVisc; /* Viscosity of liquid, cp */
    float PartDens; /* Density of particle, g/cm3 */
    float d; /* Particle Diameter, microns */
    float SetVel; /* Settling Velocity, m3/m2-h */
    float Depth; /* Depth of Clarifier, m */
    float Time; /* Detention Time, h */
    float Area; /* Clarifier area, m2 */
    float Diam; /* Diameter of circular Clarifier, m */
    float Length; /* Length of rectangular Clarifier, m */
    float Width; /* Width of rectangular Clarifier, m */
    float Volume; /* Volume of Clarifier, m3 */
    float L_W_Ratio; /* Ratio Length/Width for rectangular Clarifier */
    float Part_Conc_In_Sludge; /* solids concentration in sludge, mg/liter */
    float RemovFrac[compMax]; /* Removal fraction for each particle component */
};

typedef struct Clarifier CLARIFIER;
typedef CLARIFIER *ClarifierPtr;

struct Flotation {
    Boolean WithoutRecycle; /* True if specified by user */
    Boolean SpecRecyRatio; /* True if specified by user */
    Boolean RatioAirSolid; /* True if specified by user */
    float RecyRatio; /* Recycle Ratio */
    float SolAir; /* Air solubility, mL/L */
    float SatFac; /* Fraction of saturation */
    float LoadRate; /* Surface loading Rate, m3/m2-hr */
    float AirSolR; /* Air to solid ratio, mL/mg */
    float Press; /* Pressure, atm */
    float AirReq; /* Air requirements, m3/hr */
    float SurArea; /* Surface area, m2 */
    float Part_Conc_In_Sludge; /* solids concentration in sludge, mg/liter */
    float FloatFrac[compMax]; /* fraction of each particle component floated */
};

typedef struct Flotation *FlotationPtr;
typedef struct Flotation FLOTATION;
typedef FLOTATION *FlotationPtr;

struct GMediaFilter {
    Boolean UserSpecTime; /* True if user specifies filtration time */
    short EqnType;      /* 1: Carmen-Cozeny  
                          2: Rose  
                          3: Fair-Hatch */
    float Flux;         /* Filtration Velocity, L/m^2-min */
    float LiqVisc;      /* Liquid Viscosity, cp */
    short NumOfLayers;  /* Number of media layers */
    float FiltTime;     /* Filtration Time, hrs */
    float MaxPresDrop;  /* Max. Allowable Pressure drop, bar */
    short K;            /* Filtration Constant for Fair-Hatch Eqn */
    float PartDiam[5];  /* Particle Diameter per layer, mm */
    float alpha[5];     /* Porosity per layer */
    float Phi[5];       /* shape factor per layer */
    float Height[5];    /* Height per layer, m */
    float CleanHeadLoss; /* Clean Bed Headloss, m H2O */
    float PressureDrop; /* Overall Pressure Drop, Bar */
    float Diam;         /* Filter Diameter, m */
    float DiamMax;      /* Max Filter Diam, m */
    float WashTime;     /* Wash time, min */
    float WashRate;     /* Wash rate, L/m^2-min */
    float RC[compMax];  /* Component Retention Coef */
};

typedef struct GMediaFilter GMEDIAFILTER;
typedef GMEDIAFILTER *GMediaFilterPtr;

struct Adsorption {
    float BindCapacity; /* Actual Carbon Binding Capacity, Kg/Kg C */
    float CarbonDens;   /* Density of Carbon, Kg/m^3 */
    float BFR[compMax]; /* Binding Fractions of contaminants */
    float LinearVelocity; /* m/sec */
    Boolean LinVelSpec; /* TRUE if linear velocity is specified */
    float L_D_Ratio;    /* Length to Diameter Ratio */
    float Length;       /* Bed Length, m */
    float Diam;         /* Bed Diameter, m */
    float Visc;         /* Fluid Viscosity, cp */
    float PartDiam;     /* Particle diameter, micron */
    float VoidFrac;     /* Void Fraction of Bed */
    float PressureDrop; /* Overall Pressure Drop, Bar */
    float OtherPresDrop; /* Pressure drop for auxiliary equipment, kPa */
    float RegenSteam;   /* 0.2 kg of steam/kg of carbon */
    float ServiceLife;  /* 40,000 h of operation */
    float CarbonPrice;  /* $3.0/kg */
};

typedef struct Adsorption ADSORPTION;
typedef ADSORPTION *AdsorptionPtr;
struct Stripper {
    char DesignComp[32]; /* name of design component */
    short m_operationMode; // Operation mode
    // 0: Diameter specified
    // 1: Overall DP specified
    // 2: DP/Z specified
    float RemovFrac[compMax]; /* frac of each comp removed */
    float atot; /* total specific area of packing material */
    float dp; /* nominal diam of packing material */
    float sigmaCrit; /* critial surface tension of packing material */
    float sigmaLiq; /* actual surface tension of liquid */
    float LiqVisc; /* visc of liquid phase, kg/m-sec */
    float GasVisc; /* visc of gaseous phase, kg/m-sec */
    float Diff_L; /* diffusivity of design comp in liquid phase, cm2/sec */
    float Diff_G; /* diffusivity of design comp in gas phase, cm2/sec */
    float Cf; /* constant depending on packing, p.196 Treybal */
    float HTU; /* height of transfer unit [m] */
    float Height; /* column height [m] */
    float Diam; /* column diameter [m] */
    float MaxDiam; /* max column diameter [m] */
    float NTU; /* num of transfer units */
    float DeltaPres; /* pressure drop, bar */
    float m_unitHeightDeltaP; /* Unit Column Height Delta Pressure Pa/m */
};

typedef struct Stripper STRIPPER;
typedef *StripperPtr;

typedef struct Neutralizer {
    float LiqVol; /* m3 */
    float TotVol; /* m3 */
    float VolRatio; /* Liquid/Total */
    float ResTime; /* h */
    float AgitRate; /* KW/m3 */
    float ReactTemp; /* °C */
    float RxnHeat; /* Heat of Reaction */
    char RxnHeatComp[32]; /* Reaction Heat Component */
    char NeutralAgent[32]; /* Neutralizing Agent */
    float Excess; /* % Excess of Neutralizing Agent */
    float Extent; /* extent of reaction */
    float Stoich[compMax]; /* stoich coefficients */
};

typedef struct Neutralizer NEUTRALIZER;
typedef *NeutralizerPtr;

typedef struct AerationBasin {
    Boolean LiqVolSpec; /* If True liquid volume is specified, else */
    float DO; /* mg/l */
    float OperTemp; /* °C */
    float LiqVol; /* m3 */
    float HydResTime; /* h */
    float YieldCoeff; /* g/g */
    float BasinVolume; /* m3 */
    float VolRatio; /* Liquid/Total */
};
float PowerInput; /* kW/m³ */
float Ext_of_Bio[compMax]; /* Extent of biodegradation */
float Ext_of_Sor[compMax]; /* Extent of sorption */
float Ext_of_Stp[compMax]; /* Extent of stripping */
}

typedef struct AerationBasin AERATION_BASIN;
typedef AERATION_BASIN *AerationBasinPtr;

struct TricklingFilter {
  Boolean NRC_Model; /* If True NRC Model is used */
  float Ext_of_Bio[compMax]; /* Extent of biodegradation */
  float Ext_of_Stp[compMax]; /* Extent of stripping */
  float Depth; /* m */
  float HydLoading; /* m³/m²-h */
  float OperTemp; /* Operating Temperature of Filter, °C */
  float WeighingFactor; /* Used in NRC Model */
  float n_constant; /* Used in Eckenfelder Model */
  float Treat_Factor; /* 1/d, Used in Eckenfelder Model */
  float theta; /* temp. correction for treatability facto */
  float OrgLoading; /* kg BOD5/d */
  float Area; /* m² */
  float Yield; /* Biomass yield, g/g */
};

typedef struct TricklingFilter TRICKLINGFILTER;
typedef TRICKLINGFILTER *TricklingFilterPtr;

struct Thickener {
  Boolean SetFluxSpec; /* TRUE if Settling Flux is User Specified */
  Boolean Circular; /* TRUE if circular design is used */
  short ModelType; /* Type of model to be used in flux calculation */
  float SetFlux; /* Settling Flux, m³/m²-h */
  float SVI; /* Sludge volume index, mL/g */
  float Param1; /* Parameter in Generic Model, m/h */
  float Param2; /* Parameter in Generic Model, mL/g */
  float Area; /* Thickener area, m² */
  float Diam; /* Diameter of circular Clarifier, m */
  float Length; /* Length of rectangular Clarifier, m */
  float Width; /* Width of rectangular Clarifier, m */
  float L_W_Ratio; /* Ratio Length/Width for rectangular Clarifier */
  float Part_Conc_In_Sludge; /* solids concentration in sludge, mg/liter */
  float RemovFrac[compMax]; /* Removal fraction for each component */
};

typedef struct Thickener THICKENER;
typedef THICKENER *ThickenerPtr;

struct AnaerobicDigestion {
  Boolean HighRate; /* If True user specifies high rate digester */
  Boolean VolumeSpec; /* If True liquid volume is specified */
  float DigestTemp; /* Digestion Temp. °C for high rate digester */
  float ActiveVol; /* Active Digester Volume, m³ */
};
typedef struct AnaerobicDigestion { 
  float TotalVol; // Total Digester Volume, m^3
  float SRT; // Solids Retention Time, d
  float Ratio; // (Active/Total) Volumes
  float PowerInput; // KW/m^3
  float UtilEff; // Waste Utilization Efficiency
  float HeatEff; // Heating Units Efficiency
  float HeatingValue; // Heating Value of digester gas, kcal/m^3
  float SolidConc; // Solid Concentration in sludge, mg/L
  float GasProduct; // Gas Production, m^3/h
  float DigestGasRequired; // Digestion Gas Required for heating, m^3/h
  float AddHeatDuty; // Additional Heating Duty, kcal/h
  float NaturalGasRequired; // Nat. Gas Required for additional heati, m^3/
  float Stoich[compMax]; // stoich coefficients
  float SludgeFrac[compMax]; // Fraction of component transferred in sludge
  Boolean GasComp[compMax]; // True if that component is a digestion gas
};

typedef struct Belt_Filter { 
  Boolean WidthSpec; /* True if specified by user */
  float Tot_Sol_Cake; /* Total solids in cake, weight percent */
  float Sludge_Rate; /* Sludge-loading rate, kg/m-h */
  float Belt_Width; /* Belt width, m */
  float Sol_Recov; /* Solids recovery, percent */
  float WashFlow; /* Washwater flowrate, m^3/m-h */
  float RemovFrac[compMax]; // Removal fraction for each particle component
};

typedef struct Belt_Filter { 
  Boolean WidthSpec; /* True if specified by user */
  float Tot_Sol_Cake; /* Total solids in cake, weight percent */
  float Sludge_Rate; /* Sludge-loading rate, kg/m-h */
  float Belt_Width; /* Belt width, m */
  float Sol_Recov; /* Solids recovery, percent */
  float WashFlow; /* Washwater flowrate, m^3/m-h */
  float RemovFrac[compMax]; // Removal fraction for each particle component
};

typedef struct Belt_Filter { 
  Boolean WidthSpec; /* True if specified by user */
  float Tot_Sol_Cake; /* Total solids in cake, weight percent */
  float Sludge_Rate; /* Sludge-loading rate, kg/m-h */
  float Belt_Width; /* Belt width, m */
  float Sol_Recov; /* Solids recovery, percent */
  float WashFlow; /* Washwater flowrate, m^3/m-h */
  float RemovFrac[compMax]; // Removal fraction for each particle component
};

typedef struct Belt_Filter { 
  Boolean WidthSpec; /* True if specified by user */
  float Tot_Sol_Cake; /* Total solids in cake, weight percent */
  float Sludge_Rate; /* Sludge-loading rate, kg/m-h */
  float Belt_Width; /* Belt width, m */
  float Sol_Recov; /* Solids recovery, percent */
  float WashFlow; /* Washwater flowrate, m^3/m-h */
  float RemovFrac[compMax]; // Removal fraction for each particle component
};

typedef struct Rotary_Dryer { 
  Boolean Direct; // Direct Heating - Indirect Heating
  float DrumDiameter; // Drum Diameter, m
  float MaxDrumDiameter; // Max. Drum Diameter, m
  float EvaporationRate; // Overall Evaporation Rate, kg solvent/m^3-h
  float DryingTemp; // Drying temperature, °C
  float L_D_Ratio; // L/D Ratio
  float DrumLength; // Drum Length, m
  float SpecHeatRequirement; // Solvent Vaporization Heat, kcal/kg solven
  float SpecPowerRequirement; // Power per unit drum area, kW/m^2
  float NetDryingGasRequirement; // Net Drying Medium Requirement,
  float EvapFrac[compMax]; // Evaporation fraction
};

typedef struct Rotary_Dryer { 
  Boolean Direct; // Direct Heating - Indirect Heating
  float DrumDiameter; // Drum Diameter, m
  float MaxDrumDiameter; // Max. Drum Diameter, m
  float EvaporationRate; // Overall Evaporation Rate, kg solvent/m^3-h
  float DryingTemp; // Drying temperature, °C
  float L_D_Ratio; // L/D Ratio
  float DrumLength; // Drum Length, m
  float SpecHeatRequirement; // Solvent Vaporization Heat, kcal/kg solven
  float SpecPowerRequirement; // Power per unit drum area, kW/m^2
  float NetDryingGasRequirement; // Net Drying Medium Requirement,
  float EvapFrac[compMax]; // Evaporation fraction
};

typedef struct Rotary_Dryer { 
  Boolean Direct; // Direct Heating - Indirect Heating
  float DrumDiameter; // Drum Diameter, m
  float MaxDrumDiameter; // Max. Drum Diameter, m
  float EvaporationRate; // Overall Evaporation Rate, kg solvent/m^3-h
  float DryingTemp; // Drying temperature, °C
  float L_D_Ratio; // L/D Ratio
  float DrumLength; // Drum Length, m
  float SpecHeatRequirement; // Solvent Vaporization Heat, kcal/kg solven
  float SpecPowerRequirement; // Power per unit drum area, kW/m^2
  float NetDryingGasRequirement; // Net Drying Medium Requirement,
  float EvapFrac[compMax]; // Evaporation fraction
};

typedef struct Spray_Dryer { 
  float DryerDiameter; // Dryer Diameter, m
  float MaxDryerDiameter; // Max. Dryer Diameter, m
  float EvaporationRate; // Overall Evaporation Rate, kg solvent/m^3-h
  float DryingTemp; // Drying temperature, °C
  float L_D_Ratio; // L/D Ratio
  float DrumLength; // Drum Length, m
  float SpecHeatRequirement; // Solvent Vaporization Heat, kcal/kg solven
  float SpecPowerRequirement; // Power per unit drum area, kW/m^2
  float NetDryingGasRequirement; // Net Drying Medium Requirement,
float DryingTemp; // Drying temperature, °C
float H_D_Ratio; // H/D Ratio
float DryerHeight; // Dryer Height, m
float SpecHeatRequirement; // Solvent Vaporization Heat, kcal/kg solvent
float SpecPowerRequirement; // Power per unit feed volume, kW/m3
float NetDryingGasRequirement; // Net Drying Medium Requirement
float EvapFrac[compMax]; // Evaporation fraction

typedef struct Spray_Dryer SPRAY_DRYER;
typedef SPRAY_DRYER *Spray_DryerPtr;

tstruct Absorption {
    char DesignComp[32]; /* name of design component */
    short m_operationMode; // Operation mode 0: Diameter specified
        // 1: Overall DP specified
        // 2: DP/Z specified
    float RemovFrac[compMax]; /* frac of each comp removed */
    float atot; /* total specific area of packing material */
    float dp; /* nominal diam of packing material */
    float sigmaCrit; /* critial surface tension of packing material */
    float sigmaLiq; /* actual surface tension of liquid */
    float LigVisc; /* visc of liquid phase, kg/m-sec */
    float GasVisc; /* visc of gaseous phase, kg/m-sec */
    float Diff_L; /* diffusivity of design comp in liquid phase, cm2/sec */
    float Diff_G; /* diffusivity of design comp in gas phase, cm2/sec */
    float Cf; /* constant depending on packing, p.196 Treybal */
    float HTU; /* height of transfer unit [m] */
    float Height; /* column height [m] */
    float Diam; /* column diameter [m] */
    float MaxDiam; /* max column diameter [m] */
    float NTU; /* num of transfer units */
    float DeltaPres; /* pressure drop, bar */
    float m_unitHeightDeltaP; /* Unit Column Height Delta Pressure Pa/m */
};

typedef struct Absorption ABSORPTION;
typedef ABSORPTION *AbsorptionPtr;

struct Cyclone {
    Boolean UserSpecEff; /* TRUE if user specifies efficiency */
    float a; // Height of the inlet cross sectional area, m
    float b; // Width of the inlet cross sectional area, m
    float D; // Cyclone body diameter, m */
    float De; // Gas outlet diameter, m */
    float h; // Height of upper cylindrical body of cyclone,
    float S; /* Outlet length, m */
    float H; /* Overall cyclone height, m */
    float B; /* Outlet duct diameter for dust, m */
    float vel; /* Inlet gas velocity, m/s */
    float visc; /* viscosity of gas, cp */
    float dens; /* density of gas, kg/m3 */
    float DP; /* Pressure drop, bar */
    float DPmax; /* Max. Allowable Pressure drop, bar */
    float Eff; /* Overall collection efficiency */
float PartDens; /* Particle Density, kg/m³ */
float RemovFrac[compMax]; /* Fraction of component removed */
float MeanPartSize[10]; /* Mean Particle Size in a Range */
float Fraction_In_Range[10]; /* Fraction of component at a given range */

typedef struct Cyclone CYCLONE;
typedef CYCLONE *CyclonePtr;

typedef struct BagHouseFilter {
  Boolean FiltrationTimeSpec; /* True if specified by user */
  short ncompart; /* number of compartments */
  float dpm; /* maximum pressure drop, Pa */
  float MaxVel; /* maximum filtering velocity, m/min */
  float ctime; /* cleaning time, min */
  float ftime; /* filtration time, min */
  float acloth; /* net cloth area, m² */
  float ke; /* effective drag coefficient, Pa-min/m */
  float ks; /* specific drag coefficient, Pa-min-m/kg */
  float RemovFrac[compMax]; /* Removal fraction for each particle component */
} BagHouseFilter;
typedef struct BagHouseFilter BAGHOUSEFILTER;
typedef BAGHOUSEFILTER *BagHouseFilterPtr;

struct E_Precipitator {
  Boolean UserSpecEff; /* True if specified by user */
  float GasVel; /* entering gas velocity, m/min */
  int Nd; /* number of ducts */
  int Ns; /* number of sections */
  float PlatHet; /* plate height, m */
  float PlatLen; /* plate length, m */
  float OvEff; /* overall efficiency */
  float A_Ratio; /* aspect ratio, number */
  float PlatSep; /* plate separation, m */
  int Np; /* number of plates */
  float dp; /* pressure drop, Pa */
  float A; /* overall collection area, m² */
  float MeanPartSize[10]; /* mean particle size, micron */
  float Fr_In_Range[10]; /* fraction in size range */
  float MVel_In_Range[10]; /* migration velocity in size range, m/min */
} E_Precipitator;
typedef struct E_Precipitator ELECT_PRECIPITATOR;
typedef ELECT_PRECIPITATOR *E_PrecipitatorPtr;
APPENDIX II

UNIT OPERATION MODELS IN ENVIROCAD
Solve_Equalizer(LINK unitPtr) {
    EqualizerPtr elqPtr;
    St_CompPtr OutputCompPtr, FeedCompPtr;
    St_LINK FeedStr, OutputStr, shead;
    St_LINK assStream[1];
    short i, StrNum;
    short StreamPoints;
    double AvgFlow[26]; // Average Inflow during the time period, m3/min
    double InfluentConc[26]; // Influent conc. during the time interval, mg/L
    double EffluentConc[26]; // Effluent conc. during time interval, mg/L
    double VolumeInBasin[26]; // Volume remaining at the end of each period, m
    double BasinVolume; // Volume of equalization basin
    double CumVolFlow; // Cumulative Volume at the basin, m3
    double difference; // Difference between avg. outflow and avg inflc
    double AvgOutFlow; // Average Output Flow, m3/h
    short NumOfIntervals; // Number of intervals
    double Duration; // Time duration of each interval
    double TotalTime; // The total time of equalization
    double PositiveDiff;
    double NegativeDiff;
    double max; // Maximum influent or effluent conc., mg/L
    double InfluentPF; // Peaking factor of Influent = max/mean
    double EffluentPF; // Peaking factor of Effluent = max/mean
    double mflow, minInflow;
    double TotalInfluentLoading, MeanInfluentConc;
    short empty_index, min_index;

    elqPtr = (EqualizerPtr) unitPtr->theModel;

    // FIND FEED AND PRODUCT STREAMS
    shead = stream_head;
    Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );
    FeedStr = assStream[0];

    shead = stream_head;
    Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );
    OutputStr = assStream[0];

    // CHECK IF FEED FLOWRATE IS ZERO
    mflow = FeedStr->MassFlow;
    if( mflow < 0.0000001 )
    {
        BioAlert(BIO_ALRT_OK, "The input mass flowrate is zero!");
    }
return;

// GET CURRENT DATA STRUCTURE VALUES
NumOfIntervals = elgPtr->NumOfIntervals;
Duration = elgPtr->InterDuration;

for (i=0; i<NumOfIntervals; i++)
{
    AvgFlow[i] = elqPtr->AvgFlow[i];
    InfluentConc[i] = elqPtr->InfluentConc[i];
}

// CALCULATE THE TOTAL EQUALIZATION TIME, min
TotalTime = NumOfIntervals*Duration;

// CALCULATE TOTAL CUMULATIVE VOLUME, m3
CumVolFlow = 0.0;
for (i=0; i<NumOfIntervals; i++)
    CumVolFlow += AvgFlow[i] * Duration;

// ASSIGN SIZE INFORMATION
unitPtr->Size = Estim_VolFlow(FeedStr);

// CHECK IF CUMULATIVE VOLUME IS ZERO
if(CumVolFlow < 1.0e-8)
{
    // MATERIAL BALANCES
    Copy_Stream( FeedStr, OutputStr );

    // ESTIMATE PURCHASE COST OF EQUALIZATION BASIN
    Cost_Equalizer( unitPtr );

    // ESTIMATE ENVIRONMENTAL PROPERTIES OF THE EXIT STREAM
    Estimate_EnviroStreamProperties(OutputStr);

    return;
}

// CALCULATE EQUALIZED OUTPUT VOLUMETRIC FLOW, m3/min
AvgOutFlow = 0.0;
for (i=0; i<NumOfIntervals; i++)
    AvgOutFlow += AvgFlow[i];

AvgOutFlow /= NumOfIntervals;

// CALCULATE THE AVERAGE INFLUENT CONCENTRATION, mg/L
TotalInfluentLoading = 0.0;
for (i=0; i<NumOfIntervals; i++)
    TotalInfluentLoading += InfluentConc[i]*AvgFlow[i]*Duration;

MeanInfluentConc = TotalInfluentLoading / CumVolFlow;

if( elgPtr->UserSpecVol ) // USER SPECIFIES EQUALIZER VOLUME
{
    BasinVolume = elqPtr->BasinVolume;
}
// CALCULATE EFFLUENT CONCENTRATION AT THE END OF EACH PERIOD, mg/L

// CALCULATE THE EFFLUENT CONCENTRATION FOR THE FIRST INTERVAL, mg/L
EffluentConc[0] = (InfluentConc[0]*Duration*AvgFlow[0] + MeanInfluentConc*BasinVolume)/(Duration*AvgFlow[0] + BasinVolume);

// CALCULATE THE EFFLUENT CONC. FOR EACH INTERVAL AFTER THE FIRST, mg/L
for (i=1; i<NumOfIntervals; i++)
    EffluentConc[i] = (InfluentConc[i]*Duration*AvgFlow[i] + EffluentConc[i-1]*BasinVolume)/(Duration*AvgFlow[i] + BasinVolume);

else
    // SYSTEM ESTIMATES EQUALIZER VOLUME

    // CALCULATE MAXIMUM POSITIVE AND NEGATIVE DIFFERENCES
    // BETWEEN THE CUMULATIVE VOLUME AND THE EQUALIZED DISCHARGE
    PositiveDiff = 0.0;
    NegativeDiff = 0.0;
    CumVolFlow = 0.0;
    for (i=0; i<NumOfIntervals; i++)
    {
        CumVolFlow += AvgFlow[i] * Duration;
        difference = CumVolFlow - AvgOutFlow*(i+1)*Duration;
        if( difference > PositiveDiff )
            PositiveDiff = difference;
        if( difference < NegativeDiff )
            NegativeDiff = difference;
    }
    BasinVolume = PositiveDiff + fabs(NegativeDiff);
    elqPfr->BasinVolume = BasinVolume;

    // FIND THE INDEX OF THE PERIOD WITH THE MINIMUM INFLUENT FLOW RATE
    min_index = 0;
    minInflow = AvgFlow[0];
    for (i=0; i<NumOfIntervals; i++)
    {
        if(AvgFlow[i] < minInflow)
        {
            minInflow = AvgFlow[i];
            min_index = i;
        }
    }

    // IDENTIFY IN WHICH TIME PERIOD THE EQUALIZER IS EMPTY
    empty_index = 0;
    for (i=0; i<NumOfIntervals; i++)
    {
        if(min_index != NumOfIntervals - 1)
        {
0.0) && (AvgFlow[i+1] > AvgFlow[i]) && (i >= min_index))
{
    empty_index = i;
    break;
}
}
else // THE MINIMUM INFLOW IS THE LAST ENTRY
{
    {
        empty_index = i;
        break;
    }
}
}

// CALCULATE VOLUME REMAINING AT THE END OF EACH PERIOD, m3
VolumeInBasin[empty_index] = 0.0;
for (i = empty_index; i < NumOfIntervals; i++)
    VolumeInBasin[i+1] = VolumeInBasin[i] + (AvgFlow[i+1] - AvgOutFlow)*Duration;
VolumeInBasin[-1] = VolumeInBasin[empty_index];
for (i = 0; i < empty_index; i++)
    VolumeInBasin[i] = VolumeInBasin[i-1] + (AvgFlow[i] - AvgOutFlow)*Duration;

// CALCULATE EFFLUENT CONCENTRATION AT THE END OF EACH PERIOD, mg/L
for (i = empty_index; i < NumOfIntervals; i++)
    EffluentConc[i+1] = (AvgFlow[i+1]*Duration*InfluentConc[i+1] + VolumeInBasin[i]*EffluentConc[i])/(AvgFlow[i+1]*Duration + VolumeInBasin[i]);
EffluentConc[-1] = EffluentConc[empty_index];
for (i = 0; i <= empty_index; i++)
    EffluentConc[i] = (AvgFlow[i]*Duration*InfluentConc[i] + VolumeInBasin[i-1]*EffluentConc[i-1])/(AvgFlow[i]*Duration + VolumeInBasin[i-1]);

// CALCULATE INFLUENT PEAKING FACTOR
max = 0.0;
for (i = 0; i < NumOfIntervals; i++)
{
    if(InfluentConc[i] > max)
        max = InfluentConc[i];
}
if(MeanInfluentConc > 1.0e-8)
    InfluentPF = max/MeanInfluentConc;
else
    InfluentPF = 1.0;

// CALCULATE EFFLUENT (EQUALIZED) PEAKING FACTOR
max = 0.0;
for (i=0; i<NumOfIntervals; i++)
{
    if(EffluentConc[i] > max)
        max = EffluentConc[i];
}

if(MeanInfluentConc > 1.0e-8)
    EffluentPF = max/MeanInfluentConc; // Mean Influent Conc = Mean Effluent
else
    EffluentPF = 1.0;

// UPDATE OUTPUT POINTERS
elqPtr->AvgOutFlow = AvgOutFlow;
elqPtr->InfluentPF = InfluentPF;
elqPtr->EffluentPF = EffluentPF;

for (i=0; i<NumOfIntervals; i++)
    elqPtr->EffluentConc[i] = EffluentConc[i];

// MATERIAL BALANCES
Copy_Stream( FeedStr, OutputStr );

// ESTIMATE PURCHASE COST OF EQUALIZATION BASIN
Cost_Equalizer( unitPtr );

// ESTIMATE ENVIRONMENTAL PROPERTIES OF THE EXIT STREAM
Estimate_EnviroStreamProperties(OutputStr);

/********************************************************************
• SOLVE CLARIFIER
*
* Date: January 1993
* Reference 1: Wastewater Engineering, Metcalf & Eddy Inc.
* Paragraphs 6.5 and 9.6 p220,472
* McCabe & Smith, McGraw-Hill
********************************************************************/

Solve_Clarifier(LINK unitPtr)
{
    ClarifierPtr clrPtr;
    St_CompPtr ClarifiedCompPtr,SludgeCompPtr, FeedCompPtr;
    St_LINK FeedStr, SludgeStr;
    St_LINK ClarifiedStr, shead;
    St_LINK assStream[2];
    short i, StrNum;
    short StreamPoints;
    double LiqVisc; // Viscosity of liquid, [cp]
    double LiqDens; // Density of liquid, [g/cm3]
    double PartDens; // Density of particle, [g/cm3]
    double d; // Particle Diameter, [microns]
    double SetVel; // Settling Velocity, [m3/m2-h]
    double Depth; // Depth of Clarifier, [m]
    double Time; // Detention Time, [h]
double Area; // Clarifier area, [m2]
double Diam; // Diameter of circular Clarifier, [m]
double Length; // Length of rectangular Clarifier, [m]
double Width; // Width of rectangular Clarifier, [m]
double Volume; // Volume of Clarifier, m3
double L_W_Ratio; // Ratio Length/Width for rectangular Clarifier
double K; // Criterion for Settling regime
double b1; // Constant for calculation of settling velocity
double n; // Constant for calculation of settling velocity
double g = 9.81; // acceleration of gravity, [m/s^2]
double mflow; // Entering mass flow rate, [kg/h]
double Q; // Entering volumetric flow rate, [m^3/h]
double num; // used in the calculation of settling velocity
double den; // used in the calculation of settling velocity
double SludgeTSS_wt_percent; // wt% of TSS in sludge stream
double Frac_of_FeedLiquid_In_Sludge; // Fraction of the feed liquid mass that goes to the sludge stream
double FeedTSS_wt_percent; // Solid Conc. in feed stream, wt%
double FeedStr_LiquidMass; // Mass of liquid in feed stream
double SludgeStr_LiquidMass; // Mass of liquid in sludge stream
double FeedStr_TSS; // Mass of TSS in feed stream,
double SludgeStr_TSS; // Mass of TSS in sludge stream,
double TSS_Removed; // Mass of TSS removed, kg/h
double SludgeStr_TotalMassFlow; // Mass flow of sludge stream, kg/h
double SludgeStr_LiquidMassFlow; // Mass flow of liquid (solvent+TDS) of sludge stream, kg/h
short biomassIndex;
double intraInSludge, extraInSludge, totalInSludge;
double intraRemovalFrac, massFlowIn;
double massFlowIn1, massFlowIn2;
double intraInSludgeBefore, extraInSludgeBefore;
double intraInSludgeAfter, extraInSludgeAfter;
double solventInSludge, totalInSludgeAfter;

clrPtr = (ClarifierPtr) unitPtr->theModel;

// FIND FEED AND PRODUCT STREAMS
shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );
FeedStr = assStream[0];

shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );

if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] ) )
{
    ClarifiedStr = assStream[0];
    SludgeStr = assStream[1];
}
else
{
    ClarifiedStr = assStream[1];
    SludgeStr = assStream[0];
}
// GET CURRENT DATA STRUCTURE VALUES
LiqVisc = (clrPtr->LiqVisc)*1.0e-3;  // Convert cp to (Kg/m sec)
PartDens = (clrPtr->PartDens)*1.0e+3;  // Convert g/cm3 to Kg/m3
d = (clrPtr->d)*1.0e-6;  // Convert microns to m
SludgeTSS_wt_percent = (clrPtr->Part_Conc_In_Sludge)/1.0e6;// mg/lt to wt%
mflow = FeedStr->MassFlow;

// CHECK IF FEED FLOWRATE IS ZERO
if( fabs(mflow) < 0.0000001 )
{
    BioAlert(BIO_ALRT_OK, "The mass flowrate of the feed stream to the 'Clarifier' is zero. Please increase it so that the program can function properly!");
    return;
}

// ESTIMATE FEED STREAM LIQUID DENSITY [kg/m3]
LiqDens = Solvent_Density(FeedStr);

// CHECK IF USER SPECIFIES OVERFLOW RATE (SETTLING VELOCITY)
if(clrPtr->OverflowSpec)
    SetVel = clrPtr->SetVel;
else  // System estimates settling velocity
{

    // CHECK IF PARTICLE DENSITY IS LOWER THAN LIQUID DENSITY
    if(PartDens < LiqDens)
    {
        BioAlert(BIO_ALRT_OK, "The density of the design particle in the 'Clarifier' is less than the liquid density! Please check your data.");
        Copy_Stream(FeedStr, ClarifiedStr);
        return;
    }

    // CHECK THE CRITERION FOR SETTLING VELOCITY
    K = d*pow((g*LiqDens*(PartDens - LiqDens)/pow(LiqVisc,2.0)),(1.0/3.0));

    // SELECT CONSTANTS FOR APPROPRIATE SETTLING VELOCITY REGIME
    if((K > 0.0) && (K <= 3.3))
    {
        b1 = 24.0;
        n = 1.0;
    }
    else if((K > 3.3) && (K <= 43.6))
    {
        b1 = 18.5;
        n = 0.6;
    }
    else if ((K > 43.6) && (K <= 2360.0))
    {
        b1 = 0.44;
        n = 0.0;
    }

    // CALCULATE SETTLING VELOCITY, m/sec
num = 4.0*g*pow(d,1.0 + n)*(PartDens - LiqDens);
den = 3.0*b1*pow(LiqVisc,n)*pow(LiqDens,1.0 - n);

SetVel = (pow((num/den),1.0/(2.0 - n)))*3600.0; // Convert m/s to m/hr
clrPtr->SetVel = SetVel;
}

// ESTIMATE ENTERING VOLUMETRIC FLOW RATE [m3/h]
Q = Estim_VolFlow( FeedStr );

// CALCULATE THE SURFACE AREA OF THE CLARIFIER, m2
Area = Q / SetVel;
clrPtr->Area = Area;

// CHECK IF USER SPECIFIES DEPTH OF CLARIFIER
if(clrPtr->DepthSpec) // User specifies depth in meters
{
    Depth = clrPtr->Depth;
    Time = Depth / SetVel ;
    clrPtr->Time = Time;
}
else // User specifies detention time in hours
{
    Time = clrPtr->Time;
    Depth = Time * SetVel;
    clrPtr->Depth = Depth;
}

// CALCULATE THE VOLUME OF THE CLARIFIER [m3]
Volume = Area * Depth;
clrPtr->Volume = Volume;

// CHECK IF CIRCULAR DESIGN IS USED
if(clrPtr->Circular)
{
    Diam = pow((4.0 * Area / PI),0.5);
clrPtr->Diam = Diam;
    clrPtr->Width = 0.0; // Set rectangular characteristics zero
    clrPtr->Length = 0.0;
}
else // Rectangular design
{
    L_W_Ratio = clrPtr->L_W_Ratio;
    Width = pow((Area/L_W_Ratio),0.5);
    Length = L_W_Ratio * Width;
    clrPtr->Width = Width;
    clrPtr->Length = Length;
    clrPtr->Diam = 0.0; // Set circular characteristics zero
}

// MASS BALANCES
Estimate_EnviroStreamProperties(FeedStr);

// GET INDEX OF BIOMASS COMP AND REMOVAL FRAC OF INTRACELLULAR COMPS
intraRemovalFrac = 0.0;
biomassIndex = GetComponentIndex( biomassComp );
if( biomassIndex != -1 )
  intraRemovalFrac = clrPtr->RemovFrac[biomassIndex];

// CONSIDER REMOVAL OF PARTICULATE AND INTRACELLULAR COMPONENTS
FeedCompPtr = FeedStr->StCompList;
SludgeCompPtr = SludgeStr->StCompList;

TSS_Removed = 0.0;
for( i=0; i<NumOfComps; i++ )
  intraRemovalFrac = MAX( intraRemovalFrac, clrPtr->RemovFrac[i] );

massFlowIn = FeedCompPtr->CompMassFlow*
    theComps[i]->TS*theComps[i]->TSS_TS_Ratio;
intraInSludge = massFlowIn * FeedCompPtr->CompLocation[0] *
    intraRemovalFrac;
extraInSludge = massFlowIn * FeedCompPtr->CompLocation[1] *
    clrPtr->RemovFrac[i];
totalInSludge = intraInSludge + extraInSludge;
SludgeCompPtr->CompMassFlow = totalInSludge;
TSS_Removed += totalInSludge;
if( totalInSludge > 0.000001 )
  { SludgeCompPtr->CompLocation[0] = intraInSludge/totalInSludge;
    SludgeCompPtr->CompLocation[1] = extraInSludge/totalInSludge; }
else
  { SludgeCompPtr->CompLocation[0] = 0.0;
    SludgeCompPtr->CompLocation[1] = 1.0;
  }
FeedCompPtr = FeedCompPtr->next;
SludgeCompPtr = SludgeCompPtr->next;
SludgeStr->MassFlow = Estim_MassFlow( SludgeStr );

// CHECK IF FEED TSS CONCENTRATION IS GREATER THAN THAT IN SLUDGE STREAM
FeedTSS_wt_percent = (FeedStr->TSS_Stream*Q/1000.0)/FeedStr->MassFlow;
if( (FeedTSS_wt_percent - SludgeTSS_wt_percent) > 1.0e-8 )
  { BioAlert(BIO_ALRT_OK, "The concentration of solids in the feed stream is
greater than the desired concentration of solids in the sludge stream.
Please check your feed stream data or increase the solids concentration
in sludge!");
    return;
  }

// ESTIMATE AMOUNT OF LIQUID (SOLVENTS + TDS) IN FEED STREAM, kg/h
FeedStr_LiquidMass = FeedStr->MassFlow - (FeedStr->TSS_Stream*Q/1000.0);

// ESTIMATE TOTAL MASS FLOW OF SLUDGE STREAM, kg/h
SludgeStr_TotalMassFlow = TSS_Removed/SludgeTSS_wt_percent;

// ESTIMATE LIQUID (SOLVENTS + TDS) IN SLUDGE STREAM
SludgeStr_LiquidMassFlow = SludgeStr_TotalMassFlow - TSS_Removed;

// ESTIMATE FRACTION OF FEED LIQUID MASS PRESENT IN SLUDGE
Frac_of_FeedLiquid_In_Sludge = SludgeStr_LiquidMassFlow/FeedStr_LiquidMass;

// MASS BALANCES FOR THE EXTRACELLULAR FRACTION OF LIQUID AND SOLUBLE COMPONENTS
FeedCompPtr = FeedStr->StCompList;
SludgeCompPtr = SludgeStr->StCompList;

for( i=0; i<NumOfComps; i++)
{
    // FEED MASS FLOW OF TDS, kg/h
    massFlowIn1 = FeedCompPtr->CompMassFlow*theComps[i]->TS*(1.0 - theComps[i]->TSS_TS_Ratio);

    // FEED MASS FLOW OF SOLVENTS, kg/h
    massFlowIn2 = FeedCompPtr->CompMassFlow*(1.0 - theComps[i]->TS);

    massFlowIn = massFlowIn1 + massFlowIn2;

    intraInSludgeBefore = SludgeCompPtr->CompMassFlow * SludgeCompPtr->CompLocation[0];
    extraInSludgeBefore = SludgeCompPtr->CompMassFlow * SludgeCompPtr->CompLocation[1];
    solventInSludge = massFlowIn * FeedCompPtr->CompLocation[1] * Frac_of_FeedLiquid_In_Sludge;
    intraInSludgeAfter = intraInSludgeBefore;
    extraInSludgeAfter = extraInSludgeBefore + solventInSludge;
    totalInSludgeAfter = intraInSludgeAfter + extraInSludgeAfter;
    SludgeCompPtr->CompMassFlow = totalInSludgeAfter;
    if( totalInSludgeAfter > 0.000001 )
    {
        SludgeCompPtr->CompLocation[0] = intraInSludgeAfter/totalInSludgeAfter;
        SludgeCompPtr->CompLocation[1] = extraInSludgeAfter/totalInSludgeAfter;
    } else
    {
        SludgeCompPtr->CompLocation[0] = 0.0;
        SludgeCompPtr->CompLocation[1] = 1.0;
    }

    FeedCompPtr = FeedCompPtr->next;
    SludgeCompPtr = SludgeCompPtr->next;
}
SludgeStr->MassFlow = Estim_MassFlow( SludgeStr );

// CALCULATE FLOWRATE AND COMPOSITION OF CLARIFIED STREAM
Subtract_Stream( FeedStr, SludgeStr, ClarifiedStr );

// ESTIMATE PURCHASE COST OF CLARIFIER
Cost_Clarifier( unitPtr );

// ESTIMATE ENVIRONMENTAL PROPERTIES OF EXITING STREAMS
Estimate_EnviroStreamProperties( ClarifiedStr );
Estimate_EnviroStreamProperties( SludgeStr );

// NEW
SludgeStr->TSS_Stream = TSS_Removed*1000.0/Estim_VolFlow( SludgeStr );

/**************************************************************************
  *  SOLVE FLOTATION
  *
  *  Date:  OCTOBER 1993
  *
  *  Reference 1: Industrial Water Pollution Control, 2nd Edition
  *      By W. Wesley Eckenfelder, McGraw-Hill, 1989
  *
  *  Reference 2: Waterwater Engineering, 3rd Edition
  *      By Metcalf and Eddy, McGraw-Hill, 1991
  *
**************************************************************************/
Solve_Flotation(LINK unitPtr)
{
    FlotationPtr fltPtr;
    St_CompPtr ClarifiedCompPtr,FloatCompPtr, FeedCompPtr;
    St_LINK FeedStr, FloatStr;
    St_LINK ClarifiedStr,shead;
    St_LINK assStream[2];
    short i, StrNum;
    short StreamPoints;

double RecyRatio;        // Recycle Ratio
double SolAir;           // Air solubility, mL/L
double SatFac;           // Fraction of saturation
double LoadRate;         // Surface loading Rate, m3/m2-hr
double AirSolR;          // Air to solid ratio, mL/mg
double Press;            // Pressure, atm
double AirReq;           // Air requirements, m3/hr
double SurArea;          // Surface area, m2
double FloatFrac[compMax]; // fraction of each component removed
double SolConc;          // solid concentration in feed, mg/L
double mflow;            // Entering mass flow rate, kg/h
double TotSol;           // Entering solid flow rate, mg/h
double Q;                // Entering volumetric flow rate, m3/h
double AmountRemoved;    // Amount removed from each component

double Tot_Sol_In_Sludge; // Flow of solids in the sludge stream, [kg/h]
double Part_Vol_In_Sludge; // Flow of solids in the sludge stream, [m³/h]
double TotVol_of_Sludge; // Volumetric flow of the sludge stream, [m³/h]
double FeedLiqVol; // Volum. flow of liquid in the feed stream, [m³/h]
double Part_Conc_In_Sludge; // Conc. of solids in sludge, kg/m³
double Tot_Sol_In_Feed; // Flowrate of solids in the feed stream, [kg/h]
double FeedSolConc; // Solid Concentration in feed stream, kg/m³

double Frac_of_FeedLiqVol_In_Sludge; // Fraction of the feed liquid volume
// that goes to the sludge stream

short biomassIndex;
double intraInSludge, extraInSludge, totalInSludge;
double intraRemovalFrac, massFlowIn;

double intraInSludgeBefore, extraInSludgeBefore;
double intraInSludgeAfter, extraInSludgeAfter;

double solventInSludge, totalInSludgeAfter;

fltPtr = (FlotationPtr) unitPtr->theModel;

/* FIND FEED AND PRODUCT STREAMS */

shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );
FeedStr = assStream[0];

shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );

if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[1] ) )
{
    ClarifiedStr = assStream[0];
    FloatStr = assStream[1];
}
else
{
    ClarifiedStr = assStream[1];
    FloatStr = assStream[0];
}

mflow = FeedStr->MassFlow;

/* CHECK IF FEED FLOWRATE IS ZERO */

if( fabs(mflow) < 0.0000001 )
{
    BioAlert(BIO_ALRT_OK, "The mass flowrate of the feed stream to the
    'Flotation Tank' is zero. Please increase it so that the program can
    function properly!");
    return;
}

/* GET CURRENT DATA STRUCTURE VALUES */

RecyRatio = fltPtr->RecyRatio;
SolAir = fltPtr->SolAir;
SatFac = fltPtr->SatFac;
LoadRate = fltPtr->LoadRate;
AirSolR = fltPtr->AirSolR;
Press = fltPtr->Press;

Part_Conc_In_Sludge = (fltPtr->Part_Conc_In_Sludge)*1.0e-3;// mg/lt to kg/m³
for(i=0; i<compMax; i++)
    FloatFrac[i] = fltPtr->FloatFrac[i];

/* ESTIMATE ENTERING VOLUMETRIC FLOW RATE, m3/h */
Q = Estim_VolFlow( FeedStr );

/* ESTIMATE FEED STREAM SOLID CONCENTRATION */
SolConc = Solid_Concentration(FeedStr)*1.0e+3; /* mg/L */

/* ESTIMATE FEED STREAM TOTAL SOLID FLOW */
TotSol = Q*SolConc*1.0e+3; /* mg/hr */

/* ESTIMATE REQUIRED SURFACE AREA, m2 */
SurArea = Q/LoadRate;

/* CHECK IF USER SPECIFIES FLOTATION (i) WITHOUT RECYCLE, (ii) WITH RECYCLE 
AND RECYCLE RATIO */
if(fltPtr->WithoutRecycle)
{
    if(fltPtr->RatioAirSolid) /* system estimates pressure */
        Press = (1.0/SatFac)*((AirSolR*SolConc)/(1.3*SolAir)) + 1.0);
    else /* system air to solid ratio */
        AirSolR = (1.3*SolAir*(SatFac*Press - 1.0))/SolConc;
}
else
{
    if(fltPtr->SpecRecyRatio)
    {
        if(fltPtr->RatioAirSolid) /* system estimates pressure */
            Press = (1.0/SatFac)*((AirSolR*SolConc)/((1.3*SolAir*RecyRatio)) + 1.0);
        else /* system estimates air to solid ratio */
            AirSolR = (1.3*SolAir*RecyRatio*(SatFac*Press - 1.0))/SolConc;
    }
    else // System estimates recycle ratio for flotation with recycle
        RecyRatio = (AirSolR*SolConc)/((1.3*SolAir*(SatFac*Press - 1.0));
}

/* ESTIMATE AIR REQUIREMENTS */
AirReq = TotSol * AirSolR * 1.0e-6; /* m3/hr */

/* UPDATE THE POINTERS */
fltPtr->SurArea = SurArea;
fltPtr->Press = Press;
fltPtr->AirSolR = AirSolR;
fltPtr->RecyRatio = RecyRatio;
fltPtr->AirReq = AirReq;

// GET INDEX OF BIOMASS COMP AND REMOVAL FRAC OF INTRACELLULAR COMPS
intraRemovalFrac = 0.0;
biomassIndex = GetComponentIndex( biomassComp );
if( biomassIndex != -1 )
    intraRemovalFrac = fltPtr->FloatFrac[biomassIndex];

    // CONSIDER REMOVAL OF PARTICULATE AND INTRACELLULAR COMPONENTS

FeedCompPtr = FeedStr->StCompList;
FloatCompPtr = FloatStr->StCompList;

for( i=0; i<NumOfComps; i++ )
{
    intraRemovalFrac = MAX( intraRemovalFrac, fltPtr->FloatFrac[i] );
    massFlowIn = FeedCompPtr->CompMassFlow;
    intraInSludge = massFlowIn * FeedCompPtr->CompLocation[0] * 
                  intraRemovalFrac;
    extraInSludge = massFlowIn * FeedCompPtr->CompLocation[1] * 
                    fltPtr->FloatFrac[i];
    totalInSludge = intraInSludge + extraInSludge;

    FloatCompPtr->CompMassFlow = totalInSludge;
    if( totalInSludge > 0.000001 )
    {
        FloatCompPtr->CompLocation[0] = intraInSludge/totalInSludge;
        FloatCompPtr->CompLocation[1] = extraInSludge/totalInSludge;
    }
    else
    {
        FloatCompPtr->CompLocation[0] = 0.0;
        FloatCompPtr->CompLocation[1] = 1.0;
    }

    FeedCompPtr = FeedCompPtr->next;
    FloatCompPtr = FloatCompPtr->next;
}

FloatStr->MassFlow = Estim_MassFlow( FloatStr );

    // FLOW RATE OF SOLIDS IN SLUDGE STREAM, kg/h
    Tot_Sol_In_Sludge = Estim_SolidMassFlow( FloatStr );

    // CHECK IF THE SOLIDS CONCENTRATION IN FEED STREAM IS GREATER THAN THE
    // DESIRED CONCENTRATION IN SLUDGE STREAM
    Tot_Sol_In_Feed = Estim_SolidMassFlow( FeedStr );
    FeedSolConc = Tot_Sol_In_Feed/Estim_VolFlow(FeedStr);

    if( (FeedSolConc - Part_Conc_In_Sludge) > 1.0e-8 )
    {
        BioAlert(BIO_ALRT_OK, "The concentration of solids in the feed stream is
        greater than the desired concentration of solids in the sludge stream.
        Please check your feed stream data !");
        return;
    }

    // VOLUMETRIC FLOW RATE OF SOLIDS IN SLUDGE STREAM, m3/h
    Part_Vol_In_Sludge = Estim_VolFlow( FloatStr );

    // TOTAL VOLUMETRIC FLOW RATE OF SLUDGE STREAM, m3/h
    TotVol_of_Sludge = Tot_Sol_In_Sludge / Part_Conc_In_Sludge;
// VOLUMETRIC FLOW RATE OF LIQUID IN FEED STREAM, m3/h
FeedLiqVol = Estim_SolventVolFlow(FeedStr);

// FRACTION OF THE FEED LIQUID THAT GOES TO THE SLUDGE STREAM
Frac_of_FeedLiqVol_In_Sludge = (TotVol_of_Sludge - Part_Vol_In_Sludge) / FeedLiqVol;

if( Frac_of_FeedLiqVol_In_Sludge > 1.0 )
    Frac_of_FeedLiqVol_In_Sludge = 1.0;

if( Frac_of_FeedLiqVol_In_Sludge < 0.0 )
    Frac_of_FeedLiqVol_In_Sludge = 0.0;

// MASS BALANCES FOR THE EXTRACELLULAR FRACTION OF LIQUID COMPONENTS
FeedCompPtr = FeedStr->StCompList;
FloatCompPtr = FloatStr->StCompList;

for( i=0; i<NumOfComps; i++ )
{
    if( fltPtr->FloatFrac[i] < 0.0000001)
    {
        intraInSludgeBefore = FloatCompPtr->CompMassFlow * 
            FloatCompPtr->CompLocation[0];
        extraInSludgeBefore = FloatCompPtr->CompMassFlow * 
            FloatCompPtr->CompLocation[1];

        solventInSludge = FeedCompPtr->CompMassFlow * 
            FeedCompPtr->CompLocation[1] * Frac_of_FeedLiqVol_In_Sludge;

        intraInSludgeAfter = intraInSludgeBefore;
        extraInSludgeAfter = extraInSludgeBefore + solventInSludge;
        totalInSludgeAfter = intraInSludgeAfter + extraInSludgeAfter;

        FloatCompPtr->CompMassFlow = totalInSludgeAfter;
        if( totalInSludgeAfter > 0.000001 )
        {
            FloatCompPtr->CompLocation[0] = 
                intraInSludgeAfter/totalInSludgeAfter;
            FloatCompPtr->CompLocation[1] = 
                extraInSludgeAfter/totalInSludgeAfter;
        }
        else
        {
            FloatCompPtr->CompLocation[0] = 0.0;
            FloatCompPtr->CompLocation[1] = 1.0;
        }
    }

    FeedCompPtr = FeedCompPtr->next;
    FloatCompPtr = FloatCompPtr->next;
}

FloatStr->MassFlow = Estim_MassFlow( FloatStr );

// CALCULATE FLOWRATE AND COMPOSITION OF CLARIFIED STREAM
Subtract_Stream(FeedStr, FloatStr, ClarifiedStr);

/* ESTIMATE PURCHASE COST OF FLOTATION TANK */
Cost_Flotation(unitPtr, Q);

/* ESTIMATE ENVIRONMENTAL PROPERTIES OF EXITING STREAMS */
Estimate_EnviroStreamProperties(ClarifiedStr);
Estimate_EnviroStreamProperties(FloatStr);

/*************************************************************/
* Solve_GMediaFilter(): estimates the size and purchase       *
* cost of granular media filter                              *
*************************************************************/
Solve_GMediaFilter( LINK unitPtr )
{
    GMediaFilterPtr gmfPtr;
    St_CompPtr ProductCompPtr, WashCompPtr, FeedCompPtr;
    St_LINK FeedStr, WashStr;
    St_LINK ProdStr, WashoutStr, shead;
    St_LINK assStream[2];
    short i, StrNum;
    short StreamPoints;
    Boolean errorFlag = FALSE;

    short NumOfFilters = 0; // Number of GMediaFilters
    double Q; // Volumetric flow rate, m³/min
    double Area; // Cross sectional area of filter bed, m²
    double AmountRemoved; // Mass deposited in filter from each component, Kg/h
    double mflow; // Entering mass flow rate, kg/h
    double wflow; // Entering wash water flow rate, kg/h
    double Flux; // Filtration Rate, liters/m² min
    double Diam; // Filter diameter, m
    double DiamMax; // Maximum allowable filter diameter, m
    double WashTime; // BackWash time of the filter, min
    double WashRate; // BackWash rate of the filter, liters/m² min
    double WashFlowrate; // Adjusted Wash Stream flow rate, m³/h
    double WashVolFlow; // Volumetric flow rate of the wash stream, m³/h
    double WashAmount; // Amount removed by the wash stream, m³
    double factor; // Adjustment factor for the wash stream
    double CycleTime; // Cycle Time, h
    double Time; // Filtration time, h

    short biomassIndex;
    double intraInWashOut, extraInWashOut, totalInWashOut;
    double intraRemovalFrac;

    gmfPtr = (GMediaFilterPtr) unitPtr->theModel;

    // FIND FEED AND PRODUCT STREAMS
    shead = stream_head;
    Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );
    StreamPoints = assStream[0]->NumOfPoints;
    if( EqualPt(assStream[0]->spoints[StreamPoints-1], unitPtr->hpoints[0]))
    {
FeedStr = assStream[0];
WashStr = assStream[1];
}
else
{
    FeedStr = assStream[1];
    WashStr = assStream[0];
}
shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );
if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] ) )
{
    ProdStr = assStream[0];
    WashoutStr = assStream[1];
}
else
{
    ProdStr = assStream[1];
    WashoutStr = assStream[0];
}
mflow = FeedStr->MassFlow/unitPtr->UtilFactor;
wflow = WashStr->MassFlow/unitPtr->UtilFactor;

// CHECK IF FEED FLOW RATE IS ZERO
if( fabs(mflow) < 0.0000001 )
{
    BioAlert(BIO_ALRT_OK, "The mass flowrate of the feed stream to the
    'Granular Media Filter' is zero. Please increase it so that the program
    can function properly !");
    return;
}

// CHECK IF WASH WATER FLOW RATE IS ZERO
if( fabs(wflow) < 0.0000001 )
{
    BioAlert(BIO_ALRT_OK, "The mass flowrate of the Wash Water stream to the
    'Granular Media Filter' is zero. Please increase it so that the program
    can function properly !");
    return;
}

// GET VALUES FROM DATA STRUCTURE
Flux = gmfPtr->Flux*60.0/1000.0;  // Convert L/m2-min to m3/m2-h
DiamMax = gmfPtr->DiamMax;
WashTime = gmfPtr->WashTime/60.0;  // Convert min to h
WashRate = gmfPtr->WashRate*60.0/1000.0;  // Convert L/m2-min to m3/m2-h

// FIND VOLUMETRIC FLOW RATE, m3/h
Q = Estim_VolFlow( FeedStr );

// CHECK IF MULTIPLE FILTERS ARE NEEDED
do
{
    NumOfFilters ++;
}
// CALCULATE CROSS SECTIONAL AREA OF FILTER BED, m^2
Area = Q / Flux/ NumOfFilters;

// CALCULATE VESSEL DIAMETER, m
Diam = pow((4.0*Area/PI),0.5);

} while(Diam > DiamMax);

// ASSIGN VALUES TO DATA STRUCTURE
gmfPtr->CleanHeadLoss = CleanBedHeadlossEstimation(unitPtr, FeedStr);
gmfPtr->Diam = Diam;
unitPtr->Number = NumOfFilters;

// FIND FILTRATION TIME
if(gmfPtr->UserSpecTime)
  Time = gmfPtr->FiltTime;
else
  Time = GMFilterTimeEstimation(unitPtr, FeedStr, &errorFlag);

if( errorFlag )
{
  BioAlert(BIO_ALRT_OK, "The numerical methods failed to calculate the filtration time of the Granular Media Filter. ");
  return;
}

// ASSIGN TIMES
unitPtr->SchedTime[0] = Time;
gmfPtr->FiltTime = Time;
CycleTime = Time + WashTime;

// ESTIMATE TOTAL PRESSURE DROP, bar
gmfPtr->PressureDrop = GMFilterPressureDrop( unitPtr, FeedStr, Time);

// ADJUST FLOW RATE OF WASH IN STREAM
WashRate *= Area;  // m^3/h
WashAmount = WashRate * WashTime;  // m^3
WashFlowrate = WashAmount / CycleTime;
WashVolFlow = Estim_VolFlow( WashStr );
factor = WashFlowrate / WashVolFlow;
Multiply_Stream(WashStr, factor);

// GET INDEX OF BIOMASS COMP AND REMOVAL FRAC OF INTRACELLULAR COMPS
intraRemovalFrac = 0.0;
biomassIndex = GetComponentIndex( biomassComp );
if( biomassIndex != -1 )
  intraRemovalFrac = gmfPtr->RC[biomassIndex];

// CONSIDER REMOVAL OF INTRACELLULAR COMPONENTS
FeedCompPtr = FeedStr->StCompList;
WashCompPtr = WashoutStr->StCompList;
for( i=0; i<NumOfComps; i++ )
{

intraRemovalFrac = MAX( intraRemovalFrac, gmfPtr->RC[i] );
intraInWashOut = FeedCompPtr->CompMassFlow * 
    FeedCompPtr->CompLocation[0] * intraRemovalFrac;
extraInWashOut = FeedCompPtr->CompMassFlow * 
    FeedCompPtr->CompLocation[1] * gmfPtr->RC[i];
totalInWashOut = intraInWashOut + extraInWashOut;

WashCompPtr->CompMassFlow = totalInWashOut;
if( totalInWashOut > 0.000001 )
{
    WashCompPtr->CompLocation[0] = intraInWashOut/totalInWashOut;
    WashCompPtr->CompLocation[1] = extraInWashOut/totalInWashOut;
}
else
{
    WashCompPtr->CompLocation[0] = 0.0;
    WashCompPtr->CompLocation[1] = 1.0;
}

FeedCompPtr = FeedCompPtr->next;
WashCompPtr = WashCompPtr->next;

WashoutStr->MassFlow = EstimMassFlow( WashoutStr );

// CALCULATE FLOWRATE AND COMPOSITION OF PRODUCT STREAM
Subtract_Stream( FeedStr, WashoutStr, ProdStr );

// CALCULATE WASHOUT STREAM
Mix2_Stream(WashStr, WashoutStr, WashoutStr);

// ESTIMATE PURCHASE COST
Cost_GMediaFilter(unitPtr);

    // ESTIMATE ENVIRONMENTAL PROPERTIES OF EXITING STREAMS
Estimate_EnviroStreamProperties(ProdStr);
Estimate_EnviroStreamProperties(WashoutStr);
}

**************************************************************************
*  Function that estimates the filtration time and
*  returns the difference between the maximum and
*  the total headloss in the granular media filter
**************************************************************************

double GMediaFilterTimeEstimation(LINK unitPtr, St_LINK FeedStr, short* errorFlag)
{
    GMediaFilterPtr gmfPtr;
    int k;
    double factor = 1.6;
    double time_left; /* Lower limit of Filtration time, h */
    double time_right; /* Upper limit Filtration time, h */
    double time_middle; /* Middle point in bisection method */
double D_left; /* Total pressure difference at time_left, bar */
double D_right; /* Total pressure difference at time_right, bar */
double D_middle; /* Total pressure difference at time_middle, bar */

gmfPtr = (GMediaFilterPtr) unitPtr->theModel;

/* APPLY BISECTION METHOD TO CALCULATE THE FILTRATION TIME */
time_left = 0.0; /* Lower limit of Filtration time, h */
time_right = 2000.0; /* Upper limit of Filtration time, h */
D_left = GMFilterPressureDrop(unitPtr, FeedStr, time_left) -
        gmfPtr->MaxPresDrop;
D_right = GMFilterPressureDrop(unitPtr, FeedStr, time_right) -
        gmfPtr->MaxPresDrop;

*errorFlag = FALSE;

for(k=0; k<= MAX_ITERATIONS; k++)
{
    if((D_left * D_right) < 0.0)
    {
        while(fabs(time_left - time_right) > 0.001)
        {
            time_middle = (time_left + time_right)/2.0;
            D_middle = GMFilterPressureDrop(unitPtr, FeedStr, time_middle)
                - gmfPtr->MaxPresDrop;
            if(D_middle*D_Ieft < 0.0)
               time_right = time_middle;
            else
            {
                time_left = time_middle;
                D_left = D_middle;
            }
        }
        break;
    }

    if(fabs(D_left) < fabs(D_right))
        D_left = GMFilterPressureDrop(unitPtr, FeedStr, time_left +=
            factor*(time_left - time_right)) - gmfPtr->MaxPresDrop
    else
        D_right = GMFilterPressureDrop(unitPtr, FeedStr, time_right +=
            factor*(time_right - time_left)) - gmfPtr->MaxPresDrop;
}

if(k>MAX_ITERATIONS)
    *errorFlag = TRUE;

return(time_middle);
* Function that estimates the total pressure drop in the GMFilter given the time of filtration in hr
* K. Abeliotis 2/18/1994
*
*****************************************************************************/

double GMFilterPressureDrop(LINK unitPtr, St_LINe FeedStr, double time)
{
GMediaFilterPtr gmfPtr;
St_CompPtr FeedCompPtr;

short i;
double BedHeight; /* Total bed height, m */
double BedVolume; /* Total bed volume, m3 */
double alpha; // Average bed porosity
double RemovPartVol; // Volume of removed particles, m3
double b; // parameter used in pressure drop correlation
double nu; // Volume of deposited particles/Unit bed volume
double h0; // Clean bed headloss, m H2O
double h; // Total bed headloss, m H2O
double g = 9.81; // m/s2
double Dens; // Feed Stream Density, kg/m3
double deltaP; // Total pressure drop, Pa (kg/m-s2)

gmfPtr = (GMediaFilterPtr) unitPtr->theModel;

   // ESTIMATE AVERAGE FILTER POROSITY
   alpha = 0.0;
   for(i=0; i<gmfPtr->NumOfLayers; i++)
   {
      alpha += gmfPtr->alpha[i];
   }
   alpha /= gmfPtr->NumOfLayers;

   // ESTIMATE BED HEIGHT, m
   BedHeight = 0.0;
   for(i=0; i<gmfPtr->NumOfLayers; i++)
   {
      BedHeight += gmfPtr->Height[i];
   }

   // ESTIMATE BED VOLUME, m3
   BedVolume = (PI*pow(gmfPtr->Diam,2.0)/4.0)*BedHeight;

   // ESTIMATE VOLUME OF DEPOSITED PARTICLES, m3
   FeedCompPtr = FeedStr->StCompList;
   RemovPartVol = 0.0;
   for( i=0; i<NumOfComps; i++)
   {
      RemovPartVol += FeedCompPtr->CompMassFlow*gmfPtr->RC[i]/Component_Density(theComps[i]->name)*time/unitPtr->Number;
      FeedCompPtr = FeedCompPtr->next;
   }

   // ESTIMATE DENSITY OF FEED STREAM, Kg/m3
   Dens = Stream_Density(FeedStr);

   // PARAMETERS FOR PRESSURE DROP CORRELATION
\[ b = \frac{\alpha}{1.0 - \alpha}; \]
\[ \nu = \frac{\text{RemovPartVol}}{\text{BedVolume}}; \]

// ESTIMATE CLEAN BED HEADLOSS, m
\[ h_0 = \text{CleanBedHeadlossEstimation}(\text{unitPtr, FeedStr}); \]

// ESTIMATE TOTAL HEADLOSS, m
\[ h = h_0 \times (1.0 + (2.0 \times b + 1.0) \times (\nu/\alpha) + \text{pow}(b+1.0,2.0) \times \text{pow}(\nu/\alpha,2.0)); \]

// CONVERT HEADLOSS TO Pa
\[ \delta P = \text{Dens} \times g \times h; \]

// CONVERT Pa to bar
\[ \delta P /= 1.0E5; \]

return (\delta P); // bar

/*****************************************************************************
*     Function that estimates the clean bed headloss                      *
*     for the GMFilter using the appropriate equations                  *
*     K. Abeliotis 2/18/1994                                            *
*****************************************************************************/

double CleanBedHeadlossEstimation(LINK unitPtr, St_LINK FeedStr)
{
    GMediaFilterPtr gmfPtr;
    short i;
    double g = 9.81;       // m/s^2
    double Phi[5];
    double PartDiam[5];
    double alpha[5];
    double Height[5];
    double K;             // Fair-Hatch Eqn Filtration Constant
    double h[5];          // Clean bed headloss for each layer
    double Re[5];         // Reynold's Number
    double Cd[5];         // Friction Factor Rose Eqn
    double f[5];          // Friction Factor Carmen-Kozeny Eqn
    double htot = 0.0;    // Total Clean bed headloss
    short NumOfLayers;
    double LiqVisc;       // Kg/m-s
    double Dens;          // Kg/m^3
    double KinemVisc;     // m^2/s
    double Flux;          // m^3/m^2-h

    gmfPtr = (GMediaFilterPtr) unitPtr->theModel;

    // GET VALUES FROM DATA STRUCTURE
    NumOfLayers = gmfPtr->NumOfLayers;
    Flux = gmfPtr->Flux/60.0/1000.0;       // Convert L/m^2-min to m^3/m^2-s
    LiqVisc = gmfPtr->LiqVisc*1.0E-3;      // Convert cp to Kg/m-s
    K = gmfPtr->K;
for(i=0; i<NumOfLayers; i++)
{
    PartDiam[i] = gmfPtr->PartDiam[i]*1.0E-3; // Convert mm to m
    Phi[i] = gmfPtr->Phi[i];
    alpha[i] = gmfPtr->alpha[i];
    Height[i] = gmfPtr->Height[i];
}

// ESTIMATE DENSITY OF FEED STREAM, Kg/m3
Dens = Stream_Density(FeedStr);

// ESTIMATE CLEAN BED HEAD LOSS FOR EACH LAYER
for(i=0; i<NumOfLayers; i++)
{
    if(gmfPtr->EqnType == 1) // Carmen-Kozeny
    {
        // Reynold's Number
        Re[i] = Phi[i]*PartDiam[i]*Flux*Dens/LiqVisc;

        // Friction Factor
        f[i] = 150.0*(1.0-alpha[i])/Re[i] + 1.75;

        // Headloss, m H2O
        h[i] = (f[i]/Phi[i])*((1.0-alpha[i])/pow(alpha[i],3.0))*(Height[i]/PartDiam[i])*(pow(Flux,2.0)/g);
    }
    else if(gmfPtr->EqnType == 2) // Rose
    {
        // Reynold's Number
        Re[i] = Phi[i]*PartDiam[i]*Flux*Dens/LiqVisc;

        // Friction Factor
        Cd[i] = 24.0/Re[i] + 3.0/pow(Re[i],0.5) + 0.34;

        // Headloss, m H2O
        h[i] = (1.067/Phi[i])*(Cd[i]/pow(alpha[i],4.0))*(Height[i]/PartDiam[i])*(pow(Flux,2.0)/g);
    }
    else // Fair-Hatch
    {
        // Kinematic Viscosity, m2/s
        KinemVisc = LiqVisc/Dens;

        // Headloss, m H2O
        h[i] = K*KinemVisc*pow(Phi[i],2.0)*pow(1.0-alpha[i],2.0)/pow(alpha[i],3.0)*pow(Height[i]/pow(PartDiam[i],2.0))*(Flux/g);
    }
    htot += h[i];
}

return(htot); // m H2O
Solve_Adsorption(LINK unitPtr)  
{  
AdsorptionPtr adsPtr;  
St_CompPtr FeedCompPtr;  
St_CompPtr WasteCompPtr;  
St_CompPtr PureCompPtr;  
St_LINK FeedStr, RegenerationStr;  
St_LINK PurifiedStr, WasteStr;  
short StreamPoints;  
St_LINK shead, assStream[2];  
short i, StrNum;  
double UtilFactor = unitPtr->UtilFactor;  

double MassFlow; /* Mass Flow, Kg/h */  
double VolFlow; /* Volume Flow, m3/h */  
double ColumnVol; /* Volume of Bed, m3 */  
double Length; /* Bed Length, m */  
double Diam; /* Bed Diameter, m */  
double Area; /* Cross Sectional Area, m2 */  

double ServiceTime; /* Breakthrough Time, h */  
double RegenTime; /* Regeneration Time, h */  
double BindCap; /* Binding Capacity, kg/Kg C */  
double Vel; /* Linear Flow Rate, m/h */  
double CarbonDens; /* Density of Carbon, Kg/m3 */  
double VoidFrac; /* Void Fraction of Bed */  
double PartDiam; /* Particle Diameter, m */  

double Dens; /* Density of Feed stream, Kg/m3 */  
double DP; /* Total Pressure Drop, cm of H2O */  
double G; /* Mass loading per unit area, Kg/m2-h */  
double term1; /* Used for pressure drop calculation */  
double term2; /* Used for pressure drop calculation */  
double term3; /* Used for pressure drop calculation */  
double term4; /* Used for pressure drop calculation */  
double Visc; /* Viscosity of feed stream carrier,cp */  

double CarbonVolume; /* Total Carbon Volume in the column, m3 */  
double CarbonMass; /* Carbon Mass Requirements, Kg/h */  
double TotalCarbonMass; /* Total Carbon Mass in bed, Kg */  
double AmountRemoved; /* Amount adsorbed, kg */  
double SolventPerCycle; /* Amount of solvent adsorbed per cycle, kg */  
double CarbonMassPerBed; /* kg */  

double RegenerationSteam;  
double OldSteamFlowrate;  
short LinearVelocity;  
short Number;  

double TotalCarbonMassMax = 50000.0; // kg - max size of a carbon column
short biomassIndex;
double intrainWashOut, extrainWashOut, totalInWashOut;
double intraRemovalFrac;

adsPtr = (AdsorptionPtr) unitPtr->theModel;

/* FIND FEED AND PRODUCT STREAMS */

shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );

StreamPoints = assStream[0]->NumOfPoints;
if( EqualPt( assStream[0]->spoints[StreamPoints-1], unitPtr->hpoints[0] ) )
{
    FeedStr = assStream[0];
    RegenerationStr = assStream[1];
}
else
{
    FeedStr = assStream[1];
    RegenerationStr = assStream[0];
}

shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );

StreamPoints = assStream[0]->NumOfPoints;
if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] ) )
{
    PurifiedStr = assStream[0];
    WasteStr = assStream[1];
}
else
{
    PurifiedStr = assStream[1];
    WasteStr = assStream[0];
}

/* GET STRUCTURE AND FEED STREAM DATA */

BindCap = adsPtr->BindCapacity;
CarbonDens = adsPtr->CarbonDens;
VoidFrac = adsPtr->VoidFrac;
Visc = adsPtr->Visc * 1.0e-3 * 3600.0; /* Convert cp to Kg/m-h */
PartDiam = adsPtr->PartDiam*1.0e-3; /* Convert mm to m */

/* CHECK IF FEED FLOWRATE IS ZERO */
MassFlow = Estim_MassFlow(FeedStr);
if( MassFlow < 0.0000001 )
{
    BioAlert(BIO_ALRT_OK, "The mass flowrate of the feed stream to the 'Adsorption Column' is zero. Please increase it so that the program can function properly ");
    return;
}

/* ESTIMATE AMOUNT OF SOLVENT ADSORBED PER CYCLE */
SolventPerCycle = 0.0;
FeedCompPtr = FeedStr->StCompList;

for (i=0; i<NumOfComps; i++)
{
    if ( adsPtr->BFR[i] > 0.000001 )
    {
        SolventPerCycle += FeedCompPtr->CompMassFlow*adsPtr->BFR[i];
    }
    FeedCompPtr = FeedCompPtr->next;
}

SolventPerCycle *= unitPtr->SchedTime[0];

/* ESTIMATE AMOUNT OF CARBON PER COLUMN */
CarbonMassPerBed = SolventPerCycle / BindCap;

Number = 1;
if ( CarbonMassPerBed > TotalCarbonMassMax )
{
    Number = (short) ceil( CarbonMassPerBed/TotalCarbonMassMax );
    CarbonMassPerBed = CarbonMassPerBed/Number;
}

Number *= 2; // CA columns are always in pairs */
unitPtr->Number = Number;

unitPtr->Size = CarbonMassPerBed;

/* ESTIMATE TOTAL CARBON VOLUME, m3 */
CarbonVolume = CarbonMassPerBed / CarbonDens;

/* ESTIMATE COLUMN VOLUME, m3 */
ColumnVol = CarbonVolume / (1.0 - VoidFrac);

/* GET FEED VOLUMETRIC FLOWRATE */
VolFlow = Estim_VolFlow(FeedStr);
VolFlow /= unitPtr->UtilFactor; // actual flowrate during process time

/* CHECK IF LINEAR VELOCITY IS SPECIFIED OR NOT */
if (!adsPtr->LinVelSpec)
{
    /* ESTIMATE COLUMN DIAMETER, m */
    Diam = pow((4.0*ColumnVol/PI/adsPtr->L_D_Ratio), 0.333);
    adsPtr->Diam = Diam;

    /* ESTIMATE COLUMN LENGTH, m */
    Length = Diam * adsPtr->L_D_Ratio;
    adsPtr->Length = Length;

    /* ESTIMATE CROSS SECTIONAL AREA */
    Area = Diam * Diam * PI / 4.0;

    /* ESTIMATE LINEAR VELOCITY */
    LinearVelocity = VolFlow / Area;
    LinearVelocity /= 3600.0; /* from m/h to m/sec */
    adsPtr->LinearVelocity = LinearVelocity;
else /* Estimate column length to diameter ratio from given linear velocity */
{
    LinearVelocity = adsPtr->LinearVelocity * 3600.0; // m/sec to m/hour

    /* ESTIMATE CROSS SECTIONAL AREA */
    Area = VolFlow/LinearVelocity;

    /* ESTIMATE COLUMN DIAMETER, m */
    Diam = pow( 4*Area/PI, 0.5);
    adsPtr->Diam = Diam;

    /* ESTIMATE COLUMN LENGTH, m */
    Length = ColumnVol/Area;
    adsPtr->Length = Length;

    /* ESTIMATE LENGTH TO DIAMETER RATIO */
    adsPtr->L_D_Ratio = Length/Diam;
}

// ESTIMATE AMOUNT OF REGENERATION STEAM AND ADJUST REGENERATION STREAM
RegenerationSteam = adsPtr->RegenSteam * CarbonMassPerBed;
RegenerationSteam /= (unitPtr->SchedTime[0] + unitPtr->SchedTime[1]);
OldSteamFlowrate = RegenerationStr->MassFlow;
if( OldSteamFlowrate > 0.000001 )
    Multiply_Stream(RegenerationStr, RegenerationSteam/OldSteamFlowrate);

/* MATERIAL BALANCES */

// GET INDEX OF BIOMASS COMP AND REMOVAL FRAC OF INTRACELLULAR COMPS
intraRemovalFrac = 0.0;
biomassIndex = GetComponentIndex( biomassComp );
if( biomassIndex != -1 )
    intraRemovalFrac = adsPtr->BFR[biomassIndex];

// CONSIDER REMOVAL OF INTRACELLULAR COMPONENTS
FeedCompPtr = FeedStr->StCompList;
WasteCompPtr = WasteStr->StCompList;
for( i=0; i<NumOfComps; i++)
{
    intraRemovalFrac = MAX( intraRemovalFrac, adsPtr->BFR[i] );
intraInWashOut = FeedCompPtr->CompMassFlow * FeedCompPtr->CompLocation[0]
* intraRemovalFrac;
extraInWashOut = FeedCompPtr->CompMassFlow * FeedCompPtr->CompLocation[1]
* adsPtr->BFR[i];
totalInWashOut = intraInWashOut + extraInWashOut;
WasteCompPtr->CompMassFlow = totalInWashOut;
if( totalInWashOut > 0.000001 )
{
    WasteCompPtr->CompLocation[0] = intraInWashOut/totalInWashOut;
    WasteCompPtr->CompLocation[1] = extraInWashOut/totalInWashOut;
}
else
{
    WasteCompPtr->CompLocation[0] = 0.0;
WasteCompPtr->CompLocation[1] = 1.0;
}
FeedCompPtr = FeedCompPtr->next;
WasteCompPtr = WasteCompPtr->next;

WasteStr->MassFlow = Estim_MassFlow(WasteStr);

// CALCULATE FLOWRATE AND COMPOSITION OF PRODUCT STREAM
Subtract_Stream(FeedStr, WasteStr, PurifiedStr);

// CALCULATE WASHOUT STREAM
Mix2_Stream(RegenerationStr, WasteStr, WasteStr);

/* ESTIMATE PRESSURE DROP THROUGH THE BED USING ERGUN EQUATION, 'kg/m-hr2' */
Dens = Solvent_Density(FeedStr);
G = MassFlow/Area/Number;
term1 = ((150.0*(1.0 - VoidFrac)*Visc)/(PartDiam*G))+1.75;
term2 = Dens/pow(G,2.0);
term3 = (PartDiam/Length);
term4 = pow(VoidFrac,3.0)/(1.0 - VoidFrac);
DP = term1/(term2*term3*term4);

/* CONVERT 'kg/m-hr2' TO 'kg/m-sec2'(pascal) AND THEN TO 'bar' */
adsPtr->PressureDrop = (DP/pow(3600.0,2.0)/1.013e5) +
(adsPtr->OtherPresDrop/1.013e2);
DP = adsPtr->PressureDrop * 1.013e5;  /* from bar to Pa */

/* ESTIMATE POWER CONSUMPTION FOR THE BLOWER, kw */
if(!unitPtr->m_isPowerSet)
   unitPtr->Power = (VolFlow/3600.0)*DP/1000.0; /* in KW */

/* ESTIMATE PURCHASE COST OF COLUMN AND ITS ACCESSORIES */
Cost_Adsorption(unitPtr);

/* ESTIMATE ENVIRONMENTAL PROPERTIES OF THE EXIT STREAMS */
Estimate_EnviroStreamProperties(PurifiedStr);
Estimate_EnviroStreamProperties(WasteStr);

/**************************************************************************
• SOLVE STRIPPING COLUMN
* Programmed by: Costas Abeliotis & DP
* Date:       August 22, 1992
* Reference 1: Treybal, ed. 3rd, Chapter 6, pp. 187-209,300-322
  pp. 207-229
* Reference 3: Onda et al,"Mass Transfer Coefficients Between
  Gas and Liquid Phases in Packed Columns",
  Journal of Ch.E of Japan,Vol.1,No 1, 1968
*/
Solve_Stripper(LINK unitPtr)
{
    StripperPtr sprPtr, GCompPtr, LCompPtr;
    St_LINK Gas_In_Str, Gas_Out_Str;
    St_LINK Lig_In_Str, Liq_Out_Str;
    St_LINK shead, assStream[2];
    short i, StrNum, StreamPoints;
    double Cf, LiqVisc, GasVisc;
    double Liq_Dens, Gas_Dens;
    double m_xAxisValue, m_yAxisValue;
    double G_Prime, L_Prime;
    double S_Area;
    double AmountRemoved;
    short designCompIndex;
    short m_operationMode;
    double m_unitHeightDeltaP;
    double m_columnHeight;
    double m_deltaP;
    double m_diameter;
    double m_diamLeft, m_diamRight, m_diamMiddle;
    double fLeft, fRight, fMiddle;
    short counter;

    sprPtr = (StripperPtr) unitPtr->theModel;

    // FIND FEED AND PRODUCT STREAMS
    shead = stream_head;
    Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );

    StreamPoints = assStream[0]->NumOfPoints;
    if( EqualPt( assStream[0]->spoints[StreamPoints-1], unitPtr->hpoints[0] ) )
    {
        Liq_In_Str = assStream[0];
        Gas_In_Str = assStream[1];
    }
    else
    {
        Liq_In_Str = assStream[1];
        Gas_In_Str = assStream[0];
    }

    shead = stream_head;
    Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );

    if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] ) )
    {
        Liq_Out_Str = assStream[0];
        Gas_Out_Str = assStream[1];
    }
    else
    {
Liq_Out_Str = assStream[1];
Gas_Out_Str = assStream[0];

// CHECK IF FEED FLOWRATE IS ZERO
if( Gas_In_Str->MassFlow < 0.0000000001 )
    return;

// GET DATA_STRUCTURE VALUES
Cf = sprPtr->Cf;
LiqVisc = sprPtr->LiqVisc / 1000.0; // from cp to kg/m-s
GasVisc = sprPtr->GasVisc / 1000.0;
m_operationMode = sprPtr->m_operationMode;

// CALCULATE MATERIAL BALANCES
Copy_Stream( Gas_In_Str, Gas_Out_Str );
Copy_Stream( Liq_In_Str, Liq_Out_Str );

GCompPtr = Gas_Out_Str->StCompList;
LCompPtr = Lig_Out_Str->StCompList;

for(i=0; i<NumOfComps; i++)
{
    AmountRemoved = LCompPtr->CompMassFlow * sprPtr->RemovFrac[i];
    LCompPtr->CompMassFlow *= 1 - sprPtr->RemovFrac[i];
    GCompPtr->CompMassFlow += AmountRemoved;
    GCompPtr = GCompPtr->next;
    LCompPtr = LCompPtr->next;
}

Gas_Out_Str->MassFlow = Estim_MassFlow( Gas_Out_Str );
Lig_Out_Str->MassFlow = Estim_MassFlow( Lig_Out_Str );

// COLUMN SIZING
Lig_Flow = Lig_In_Str->MassFlow / 3600; // from kg/h to kg/sec
Gas_Flow = Gas_In_Str->MassFlow / 3600;
Lig_Dens = Stream_Density( Liq_In_Str);
Gas_Dens = Stream_Density( Gas_In_Str );

// FIND THE VALUE OF THE X-AXIS IN FIGURE 6.34 TREYBAL
m_xAxisValue = Lig_Flow/Gas_Flow * pow( Gas_Dens/(Lig_Dens - Gas_Dens), 0.5 );

if(m_operationMode == 0) // Column Diameter specified
{
    // GET DIAMETER FROM DATA STRUCTURE
    m_diameter = sprPtr->Diam;
    S_Area = PI*pow(m_diameter, 2.0)/4.0;
    G_Prime = Gas_Flow / S_Area;

    // FIND THE VALUE OF THE Y-AXIS IN FIGURE 6.34 TREYBAL
    m_yAxisValue = pow(G_Prime, 2.0)*Cf*pow(LiqVisc, 0.1)/
                   (Gas_Dens*(Lig_Dens - Gas_Dens));
// CALCULATE UNIT HEIGHT PRESSURE DROP
sprPtr->m_unitHeightDeltaP = GetUnitHeightDeltaP(m_xAxisValue, m_yAxisValue);

// CALCULATE COLUMN HEIGHT
sprPtr->Height = CalculateStripperHeight(unitPtr, m_diameter);

// CALCULATE TOTAL PRESSURE DROP
sprPtr->DeltaPres = CalculateStripperPressureDrop(unitPtr, m_diameter);

} else if(m_operationMode == 2) // DP/Z specified
{

// GET DIAMETER FROM DATA STRUCTURE
m_unitHeightDeltaP = sprPtr->m_unitHeightDeltaP;

// CALCULATE Y-AXIS VALUE
m_yAxisValue = GetYAxisValue(m_xAxisValue, m_unitHeightDeltaP);

G_Prime = sqrt(m_yAxisValue*Gas_Dens*(Liq_Dens - Gas_Dens) / (Cf*pow(LiqVisc, 0.1)));

// CALCULATE COLUMN DIAMETER
m_diameter = sqrt(4.0*Gas_Flow / (PI*G_Prime));
sprPtr->Diam = m_diameter;

// CALCULATE COLUMN HEIGHT
sprPtr->Height = CalculateStripperHeight(unitPtr, m_diameter);

// CALCULATE TOTAL PRESSURE DROP
sprPtr->DeltaPres = CalculateStripperPressureDrop(unitPtr, m_diameter);
}
else // Total DP specified
{

// GET THE GIVEN TOTAL PRESSURE DROP
m_deltaP = sprPtr->DeltaPres;

// USE BISECTION METHOD TO FIND THE DIAMETER THAT MATCHES THE GIVEN DP
m_diamLeft = 0.001;
m_diamRight = 50.0;
fLeft = CalculateStripperPressureDrop(unitPtr, m_diamLeft) - m_deltaP;
fRight = CalculateStripperPressureDrop(unitPtr, m_diamRight) - m_deltaP;

counter = 0;
if( fLeft * fRight < 0 )
{
    while (counter<300 && fabs(m_diamLeft - m_diamRight) > 0.001)
    {
        m_diamMiddle = (m_diamLeft + m_diamRight)/2.0;
        fMiddle = CalculateStripperPressureDrop(unitPtr, m_diamMiddle) - m_deltaP;
        if (fMiddle * fLeft < 0)
            m_diamRight = m_diamMiddle;
        else
{  
  m_diamLeft = m_diamMiddle;  
  fLeft = fMiddle;  
}

counter++;

else{
  BioAlert(BIO_ALRT_OK, "No column diameter value could be found in the range [0,50] m ");
  return;
}

// WHEN SOLUTION REACHED  
// WHEN SOLUTION REACHED
m_diameter = m_diamMiddle;

S_Area = PI*pow(m_diameter, 2.0)/4.0;
G_Prime = Gas_Flow / S_Area;

// FIND THE VALUE OF THE Y-AXIS IN FIGURE 6.34 TREYBAL
m_yAxisValue = pow(G_Prime, 2.0)*Cf*pow(LiqVisc, 0.1)/
                    (Gas_Dens*(Liq_Dens - Gas_Dens));

// CALCULATE UNIT HEIGHT PRESSURE DROP
sprPtr->m_unitHeightDeltaP = GetUnitHeightDeltaP(m_xAxisValue, m_yAxisValue);

// CALCULATE COLUMN HEIGHT
sprPtr->Height = CalculateStripperHeight(unitPtr, m_diameter);

}  

// ESTIMATE PURCHASE COST OF STRIPPING COLUMN
Cost_Stripper(unitPtr);

// ESTIMATE ENVIRONMENTAL PROPERTIES OF EXITING STREAMS
Estimate_EnviroStreamProperties(Liq_Out_Str);
Estimate_EnviroStreamProperties(Gas_Out_Str);

}
double CalculateStripperPressureDrop(LINK unitPtr, double m_diameter)
{
  StripperPtr    sprPtr;
  St_CompPtr     GCompPtr, LCompPtr;
  St_LINK        Gas_In_Str, Gas_Out_Str;
  St_LINK        Liq_In_Str, Liq_Out_Str;
  St_LINK        shead, assStream[2];
  short          i, StrNum, StreamPoints;
  double         LiqVisc, Cf;
  double         Liq_Flow, Gas_Flow;
  double         Liq_Dens, Gas_Dens;
  double         m_xAxisValue, m_yAxisValue;
  double         G_Prime;
  double         S_Area;
double m_columnHeight;
double m_unitHeightDeltaP;

sprPtr = (StripperPtr) unitPtr->theModel;

// FIND FEED AND PRODUCT STREAMS
shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );

StreamPoints = assStream[0]->NumOfPoints;
if( EqualPt( assStream[0]->spoints[StreamPoints-1], unitPtr->hpoints[0] ) )
{
    Liq_In_Str = assStream[0];
    Gas_In_Str = assStream[1];
}
else
{
    Liq_In_Str = assStream[1];
    Gas_In_Str = assStream[0];
}

shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );

if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] ) )
{
    Liq_Out_Str = assStream[0];
    Gas_Out_Str = assStream[1];
}
else
{
    Liq_Out_Str = assStream[1];
    Gas_Out_Str = assStream[0];
}

// GET DATA_STRUCTURE VALUES
LiqVisc = sprPtr->LiqVisc / 1000.0;  // from cp to kg/m-s
Cf = sprPtr->Cf;

// GET STREAM DENSITIES
Liq_Dens = Stream_Density( Liq_In_Str);
Gas_Dens = Stream_Density( Gas_In_Str );

// CALCULATE G_Prime
Liq_Flow = Liq_In_Str->MassFlow / 3600;
Gas_Flow = Gas_In_Str->MassFlow / 3600;
S_Area = PI*pow(m_diameter, 2.0)/4.0;
G_Prime = Gas_Flow / S_Area;

// FIND THE VALUE OF THE X-AXIS IN FIGURE 6.34 TREYBAL
m_xAxisValue = Liq_Flow/Gas_Flow * pow( Gas_Dens/(Liq_Dens - Gas_Dens), 0.5 );

// FIND THE VALUE OF THE Y-AXIS IN FIGURE 6.34 TREYBAL
m_yAxisValue = pow(G_Prime, 2.0)*Cf*pow(LiqVisc, 0.1)/
    (Gas_Dens*(Liq_Dens - Gas_Dens));

// CALCULATE UNIT HEIGHT PRESSURE DROP
m_unitHeightDeltaP = GetUnitHeightDeltaP(m_xAxisValue, m_yAxisValue);

// CALCULATE COLUMN HEIGHT
m_columnHeight = CalculateStripperHeight(unitPtr, m_diameter);

return (m_unitHeightDeltaP*m_columnHeight);
}

double CalculateStripperHeight(LINK unitPtr, double m_diameter)
{
  StripperPtr sprPtr;
  St_CompPtr GCompPtr, LCompPtr;
  St_LINK Gas_In_Str, Gas_Out_Str;
  St_LINK Liq_In_Str, Liq_Out_Str;
  St_LINK shead, assStream[2];
  short i, StrNum, StreamPoints;
  double var1;
  double LiqVisc, GasVisc;
  double dp, atot, awet, kL, kG, KLa;
  double Re, Fr, We;
  double sigmaLiq, sigmaCrit;
  double Diff_L, Diff_G, m;
  double Liq_Flow, Gas_Flow;
  double Liq_Dens, Gas_Dens;
  double m_xAxisValue, m_yAxisValue;
  double G_Prime, L_Prime;
  double S_Area;
  double HTU, NTU;
  double A_Prime, S_Factor;
  double y1, x1, x2;
  double R = 82.06e-6; // Universal gas constant, m3-atm/mole-K
  short designCompIndex;
  double m_unitHeightDeltaP;

  sprPtr = (StripperPtr) unitPtr->theModel;

  // FIND FEED AND PRODUCT STREAMS
  shead = stream_head;
  Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );

  StreamPoints = assStream[0]->NumOfPoints;
  if( EqualPt( assStream[0]->spoints[StreamPoints-1], unitPtr->hpoints[0] ) )
  {
    Liq_In_Str = assStream[0];
    Gas_In_Str = assStream[1];
  }
  else
  {
    Liq_In_Str = assStream[1];
    Gas_In_Str = assStream[0];
  }

  shead = stream_head;
  Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );
if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] ) )
{
    Liq_Out_Str = assStream[0];
    Gas_Out_Str = assStream[1];
}
else
{
    Liq_Out_Str = assStream[1];
    Gas_Out_Str = assStream[0];
}

// GET DATA_STRUCTURE VALUES
LiqVisc  = sprPtr->LiqVisc / 1000.0; // from cp to kg/m-s
GasVisc  = sprPtr->GasVisc  / 1000.0;
atot     = sprPtr->atot;
SigmaCrit = sprPtr->sigmaCrit/1000.0; // from dyn/cm to N/m
SigmaLiq  = sprPtr->sigmaLiq/1000.0; // from dyn/cm to N/m
dp        = sprPtr->dp;
Diff_L    = sprPtr->Diff_L   / 10000.0; // from cm2/sec to m2/sec
Diff_G    = sprPtr->Diff_G   / 10000.0;

// GET HENRY'S LAW CONSTANT FOR DESIGN COMPONENT
designCompIndex = GetComponentIndex(sprPtr->DesignComp);
m = theComps[designCompIndex]->Henry;
m /= R*298.15; // Convert Henry's constant from atm-m3/mole to dimensionles

// CALCULATE G_Prime, L_Prime
Liq_Flow = Liq_In_Str->MassFlow / 3600; // from kg/h to kg/sec
Gas_Flow = Gas_In_Str->MassFlow / 3600;

Liq_Dens = Stream_Density( Lig_In_Str);
Gas_Dens = Stream_Density( Gas_In_Str );
S_Area = PI*pow(m_diameter, 2.0)/4.0;
L_Prime = Liq_Flow / S_Area;
G_Prime = Gas_Flow / S_Area;

// ESTIMATE COLUMN HEIGHT
Re = L_Prime/atot/LiqVisc;
Fr = (pow(L_Prime,2)*atot)/(pow(Liq_Dens,2)*9.81);
We = pow(L_Prime,2)/(Liq_Dens*Dens*sigmaLiq*atot);

// ONDA CORRELATIONS
awet = atot*(1-exp(-1.45*pow((sigmaLiq/sigmaCrit),0.75)*
pow(Re,0.1)*pow(Fr,-0.05)*pow(We,0.2))));

kG = atot*Diff_G*5.23*pow((G_Prime/atot/GasVisc),0.7)*
pow((GasVisc/Gas_Dens/Diff_G),0.33)*pow((atot*dp),-2);
kL = (0.0051*pow((L_Prime/awet/LiqVisc),0.66)*pow((LiqVisc/Liq_Dens/Diff_L),
-0.5)*pow((atot*dp),-0.4))/pow((Liq_Dens/LiqVisc/9.81),0.33);

KLa = 1.0 / ( 1/(kG*awet) + m/(kL*awet) );

HTU = L_Prime/Liq_Dens/KLa; // Height of an stripping transfer unit

// ESTIMATE NUMBER OF TRANSFER UNITS
Liq_Flow = Estim_MoleFlow( Liq_In_Str );
Gas_Flow = Estim_MoleFlow( Gas_In_Str );

A_Factor = Liq_Flow / m / Gas_Flow;
S_Factor = 1/A_Factor;

if( S_Factor < 1.0 )
{
    BioAlert(BIO_ALRT_OK, "Please increase the gas flowrate or reduce the liquid flowrate of the stripper so that the unit can function properly!");
    S_Factor = 1.00000001;
}

y1 = Comp_MoleFrac( Gas_In_Str, sprPtr->DesignComp );
x1 = Comp_MoleFrac( Liq_Out_Str, sprPtr->DesignComp );
x2 = Comp_MoleFrac( Liq_In_Str, sprPtr->DesignComp );

if((x1 - y1/m) < 0.0)
    NTU = 0.0;
else
{
    var1 = ((x2-y1/m)/(x1-y1/m))*(1-1/S_Factor)+(1/S_Factor);
    NTU = log(var1)/(1 - 1/S_Factor);
}

// ESTIMATE COLUMN HEIGHT
return (NTU * HTU);

/*
	 Function that correlates the lines of Fig. 6-34 of Treybal 3rd ed.
	 Function that returns DP/Z in Pa/m
	 */

double GetUnitHeightDeltaP(double m_xAxisValue, double m_yAxisValue)
{
    int i, m_matchIndex;
double m_unitHeightDeltaP;
double m_yMax; // y Max for the given m_xAxisValue
double m_yMin; // y Min for the given m_xAxisValue
double m_y[7];
double m_dP[7];

    // MAKE SURE THAT THE m_xAxisValue IS WITHIN THE LIMITS OF FIG. 6.34
    if(m_xAxisValue < 0.01)
        m_xAxisValue = 0.01;
    if(m_xAxisValue > 10.0)
        m_xAxisValue = 10.0;

    // GAS PRESSURE DROP = 50 Pa/m
    m_dP[0] = 50.0;
    m_y[0] = 3.7446e-3 - 5.7794e-3*log10(m_xAxisValue);
// GAS PRESSURE DROP = 100 Pa/m
m_dP[1] = 100.0;
m_y[1] = 8.16e-3 - 1.2956e-2*10g10(m_xAxisValue);

// GAS PRESSURE DROP = 200 Pa/m
m_dP[2] = 200.0;
m_y[2] = 1.336e-2 - 2.4435e-2*10g10(m_xAxisValue);

// GAS PRESSURE DROP = 400 Pa/m
m_dP[3] = 400.0;
m_y[3] = 1.9603e-2 - 4.2356e-2*10g10(m_xAxisValue);

// GAS PRESSURE DROP = 800 Pa/m
m_dP[4] = 800.0;
m_y[4] = 2.8661e-2 - 7.0711e-2*10g10(m_xAxisValue);

// GAS PRESSURE DROP = 1200 Pa/m
m_dP[5] = 1200.0;
m_y[5] = 2.7198e-2 - 9.914e-4*10g10(m_xAxisValue);

// GAS PRESSURE DROP = 2200 Pa/m - APPROXIMATE FLOODING
m_dP[6] = 2200.0;
m_y[6] = 3.7391e-2 - 0.16267*10g10(m_xAxisValue);

// FIND MAX AND MIN VALUES OF Y FOR THE GIVEN X
m_yMax = 3.7391e-2 - 0.16267*10g10(m_xAxisValue);   // Flooding Line
m_yMin = 3.7446e-3 - 5.7794e-3*10g10(m_xAxisValue);   // Line for DP = 50 Pa/m

// CHECK IF m_yAxisValue is BOUNDED
if((m_yAxisValue >= m_yMin) && (m_yAxisValue < m_yMax))
{
    // TRY TO LOCATE Y-AXIS VALUE
    m_matchIndex = 0;
    for(i=0; i<7; i++)
    {
        if((m_yAxisValue >= m_y[i]) && (m_yAxisValue < m_y[i+1]))
        {
            m_matchIndex = i;
            break;
        }
    }

    // FIND DP CORRESPONDING TO THE Y-AXIS VALUE
    m_unitHeightDeltaP = m_dP[m_matchIndex] +((m_yAxisValue - m_y[m_matchIndex])/
                  (m_y[m_matchIndex+1] - m_y[m_matchIndex]))*
                     (m_dP[m_matchIndex+1] - m_dP[m_matchIndex]);
}
else
{
    if(m_yAxisValue < m_yMin && m_yAxisValue > 0.0015)
        m_unitHeightDeltaP = m_dP[0] +((m_yAxisValue - m_y[0])/
                      (m_y[0] - 0.0015))*(m_dP[0] - 0.0);
    else if(m_yAxisValue < 0.0015)
        m_unitHeightDeltaP = 0.0;
    else if(m_yAxisValue > m_yMax)
        m_unitHeightDeltaP = m_dP[6];
}
double GetYAxisValue(double m_AxisValue, double m_unitHeightDeltaP)
{
    int i, m_matchIndex;
    double m_yAxisValue;
    double m_yMax;  // y Max for the given m_xAxisValue
    double m_yMin;  // y Min for the given m_xAxisValue
    double m_y[7];
    double m_dP[7];

    // MAKE SURE THAT THE m_xAxisValue IS WITHIN THE LIMITS OF FIG. 6.34
    if(m_xAxisValue < 0.01)
        m_xAxisValue = 0.01;
    if(m_xAxisValue > 10.0)
        m_xAxisValue = 10.0;

    // GAS PRESSURE DROP = 50 Pa/m
    m_dP[0] = 50.0;
    m_y[0] = 3.7446e-3 - 5.7794e-3*log10(m_xAxisValue);

    // GAS PRESSURE DROP = 100 Pa/m
    m_dP[1] = 100.0;
    m_y[1] = 8.16e-3 - 1.2956e-2*log10(m_xAxisValue);

    // GAS PRESSURE DROP = 200 Pa/m
    m_dP[2] = 200.0;
    m_y[2] = 1.336e-2 - 2.4435e-2*log10(m_xAxisValue);

    // GAS PRESSURE DROP = 400 Pa/m
    m_dP[3] = 400.0;
    m_y[3] = 1.9603e-2 - 4.2356e-2*log10(m_xAxisValue);

    // GAS PRESSURE DROP = 800 Pa/m
    m_dP[4] = 800.0;
    m_y[4] = 2.8661e-2 - 7.0711e-2*log10(m_xAxisValue);

    // GAS PRESSURE DROP = 1200 Pa/m
    m_dP[5] = 1200.0;
    m_y[5] = 2.7198e-2 - 9.914e-3*log10(m_xAxisValue);

    // GAS PRESSURE DROP = 2200 Pa/m - APPROXIMATE FLOODING
    m_dP[6] = 2200.0;
    m_y[6] = 3.7391e-2 - 0.16267*log10(m_xAxisValue);

    // FIND MAX AND MIN VALUES OF Y FOR THE GIVEN X
    m_yMax = 3.7391e-2 - 0.16267*log10(m_xAxisValue);  // Flooding Line
m_yMin = 3.7446e-3 - 5.7794e-3*log10(m_xAxisValue); // Line for DP = 50 Pa/m

// TRY TO LOCATE THE GIVEN DP VALUE
m_matchIndex = 0;
for(i=0; i<7; i++)
{
    if((m_unitHeightDeltaP >= m_dP[i]) && (m_unitHeightDeltaP < m_dP[i+1]))
    {
        m_matchIndex = i;
        break;
    }
}

// FIND Y-AXIS VALUE CORRESPONDING TO THE GIVEN DP VALUE
m_yAxisValue = m_y[m_matchIndex] +((m_unitHeightDeltaP - m_dP[m_matchIndex])/
    (m_dP[m_matchIndex+1] -  m_dP[m_matchIndex]))*
    (m_y[m_matchIndex+1] - m_y[m_matchIndex]); 
return(m_yAxisValue);

/**********************************************************/
* Solve_Neutralizer(): estimates the size and purchase     *
*                        cost of a Neutralizer            *
**********************************************************/
Solve_Neutralizer(LINK unitPtr)
{
    NeutralizerPtr ntrPtr;
    St_LINK strFeed, strProd, strNeutral;
    St_LINK shead;
    St_LINK assStream[2];
    short i, Number, StreamPoints;
    short StrNum;

double FeedMassFlow; // kg/h
double LiqVol, TotVol; // m3
double ReacTemp; // C
double VolRatio;
double AgitRate; // KW/m3
double AgiPower; // KW
double UtilFactor;
double Excess; // % mass excess of neutralizing agent

    St_CompPtr FCompPtr, PCompPtr;
    double Extent, pivot;
    double mFlowOut[compMax];
    double DeltaMass;
    double mFlowKey, mFlowIn;
    double Ai, Ak,Ratio_Max;
    double Intra, Extra;
    double ReactStoichAmount,NeutralStoichAmount;
    double StoichRatio,factor,ReactFlowIn,NeutralAgentIn;
    short index;
    short NeutralAgentIndex;
St_CompPtr RxnCompPtrIn1, RxnCompPtrIn2;
St_CompPtr RxnCompPtrOut;
char RxnCompName[32];
double RxnCompIn, RxnCompOut, RxnCompMFlow;
double Duty, dens;
double BatchTime, BatchVolume;
short NumOfCycles;

ntrPtr = (NeutralizerPtr) unitPtr->theModel;

// FIND FEED AND PRODUCT STREAMS
shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );
StreamPoints = assStream[0]->NumOfPoints;
if( EqualPt( assStream[0]->spoints[StreamPoints-1], unitPtr->hpoints[0] ) ) {
strNeutral = assStream[0];
strFeed = assStream[1];
}
else {
strNeutral = assStream[1];
strFeed = assStream[0];
}

shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );
strProd = assStream[0];

if( strFeed->MassFlow < 0.0000001 ) {
BioAlert(BIO_ALRT_OK, "The mass flowrate of the feed stream to the
'Neutralizer' is zero. Please increase it so that the program can
function properly !");
return;
}

if( strNeutral->MassFlow < 0.0000001 ) {
BioAlert(BIO_ALRT_OK, "The mass flowrate of the neutralization stream
to the 'Neutralizer' is zero. Please increase it so that the program
can function properly !");
return;
}

// GET COMMON DATA STRUCTURE INFORMATION
VolRatio = ntrPtr->VolRatio;
AgitRate = ntrPtr->AgitRate;
Extent = ntrPtr->Extent;
ReactTemp = ntrPtr->ReactTemp;
Excess = ntrPtr->Excess/100.0;
UtilFactor = unitPtr->UtilFactor;

// IDENTIFY NEUTRALIZATION AGENT
NeutralAgentIndex = GetComponentIndex( ntrPtr->NeutralAgent );

// CHECK IF ENTERING AMOUNT FOR NEUTRALIZING AGENT HAS BEEN SPECIFIED
FCompPtr = strNeutral->StCompList;
NeutralAgentIn = 0.0;
for(i=0; i<NumOfComps; i++)
{
    if( i == NeutralAgentIndex)
    {
        NeutralAgentIn = FCompPtr->CompMassFlow;
        break;
    }
}

// CALCULATE TOTAL AMOUNT OF ENTERING REACTANTS (OTHER THAN NEUTR. AGENT, kg/h)
FCompPtr = strFeed->StCompList;
ReactFlowIn = 0.0;
for(i=0; i<NumOfComps; i++)
{
    if( (ntrPtr->Stoich[i] < -0.0000001) && (i != NeutralAgentIndex))
        ReactFlowIn += FCompPtr->CompMassFlow;
}

// CALCULATE ADJUSTMENT FACTOR
if(ReactFlowIn < 1.0e-6)
{
    BioAlert(BIO_ALRT_OK, "The entering mass of the reactants in the 'Neutralizer' is zero. Please check your entering stream data !");
    return;
}

// CALCULATE STOICHIOMETRIC AMOUNT OF REACTANTS (OTHER THAN NEUTR. AGENT, kg/h)
ReactStoichAmount = 0.0;
for(i=0; i<NumOfComps; i++)
{
    if( (ntrPtr->Stoich[i] < -0.0000001) && (i != NeutralAgentIndex))
        ReactStoichAmount += fabs(ntrPtr->Stoich[i]);
}

// CALCULATE STOICHIOMETRIC AMOUNT OF NEUTRALIZING AGENT, kg/h
NeutralStoichAmount = fabs(ntrPtr->Stoich[NeutralAgentIndex]);

// kg OF AGENT PER kg OF REACTANT
StoichRatio = NeutralStoichAmount/ReactStoichAmount;
// CALCULATE ADJUSTMENT FACTOR
factor = 1.0;
if (NeutralAgentIn > 0.00000000001)
factor = StoichRatio*ReactFlowIn*(1.0 + Excess)/NeutralAgentIn;

// ESTIMATE ADJUSTED NEUTRALIZING STREAM
Multiply_Stream(strNeutral, factor);

// MIX THE TWO STREAMS AND CREATE A COMPOSITE FEED STREAM
Mix2_Stream(strFeed, strNeutral, stream_dummy);

// CHECK IF FEED FLOWRATE IS ZERO
FeedMassFlow = stream_dummy->MassFlow;

// ESTIMATE THE VOLUME OF MATERIAL PROCESSED PER BATCH
BatchTime = oper_modePtr->PlantBatchTime;
NumOfCycles = unitPtr->SchedTime[4];
BatchVolume = BatchTime * FeedMassFlow / 1000.0; // m³/batch
BatchVolume /= NumOfCycles;

// CONSIDER THE EFFECT OF THE UtilFactor
FeedMassFlow /= UtilFactor;

// BATCH OPERATION
dens = Stream_Density(stream_dummy);
if (unitPtr->BatchOperation)
{
    if (unitPtr->m_isRatingMode) // RATE MODE OF CALCULATION
        LiqVol = ntrPtr->LiqVol;
    else
        LiqVol = BatchVolume;

    ntrPtr->LiqVol = LiqVol;
}
else // CONTINUOUS OPERATION
{
    if (unitPtr->m_isRatingMode) // RATE MODE OF CALCULATION
    {
        LiqVol = ntrPtr->LiqVol;
        ResTime = LiqVol / FeedMassFlow * dens; // in h
        ntrPtr->ResTime = ResTime;
    }
    // DESIGN MODE OF CALCULATION
    {
        ResTime = ntrPtr->ResTime;
        LiqVol = FeedMassFlow * ResTime / dens; // in m³
        ntrPtr->LiqVol = LiqVol;
    }
}

// ESTIMATE TOTAL VOLUME OF VESSEL
TotVol = LiqVol / VolRatio;
ntrPtr->TotVol = TotVol;

// ASSIGN VALUE TO SIZE VARIABLE
unitPtr->Size = TotVol;

// FIND REACTANT OF SMALLEST (ABS LARGEST) STOICH COEFFICIENT
PCompPtr = stream_dummy->StCompList;
pivot = 0.0;
Ratio_Max = -32000000.0;

for(i=0; i<NumOfComps; i++)
{
 if( ntrPtr->Stoich[i] < pivot )
 {
     pivot = ntrPtr->Stoich[i];
     Ratio_Max = PCompPtr->ComoMassFlow/ntrPtr->Stoich[i];
 }
 PCompPtr = PCompPtr->next;
}

PCompPtr = stream_dummy->StCompList;

for(i=0; i<NumOfComps; i++)
{
 if( ntrPtr->Stoich[i] < -0.0000001 )
 {
     mFlowIn = PCompPtr->CompMassFlow;
     if( mFlowIn/ntrPtr->Stoich[i] >= Ratio_Max )
     {
         Ratio_Max = mFlowIn/ntrPtr->Stoich[i];
         Ak = ntrPtr->Stoich[i];
         mFlowKey = PCompPtr->CompMassFlow;
     }
 }
 PCompPtr = PCompPtr->next;
}

// ESTIMATE OUTLET MASS FLOWS BASED ON THE ABOVE KEY COMPONENT
continue_1:
FCompPtr = stream_dummy->StCompList;
for(i=0; i<NumOfComps; i++)
{
 mFlowIn = FCompPtr->CompMassFlow;
 Ai = ntrPtr->Stoich[i];
 mFlowOut[i] = mFlowIn - mFlowKey * ( Ai / Ak * Extent );
 FCompPtr = FCompPtr->next;
}

// CHECK IF ANY OUTLET FLOW IS NEGATIVE AND IF YES FIND THE MINIMUM
pivot = 0.001;
FCompPtr = stream_dummy->StCompList;
for(i=0; i<NumOfComps; i++)
{
 if( mFlowOut[i] < 0.0 && mFlowOut[i] < pivot )
 {
     pivot = mFlowOut[i];
     mFlowKey = FCompPtr->CompMassFlow;
     Ak = ntrPtr->Stoich[i];
 }
 FCompPtr = FCompPtr->next;
if( pivot < 0.0 )  // there is at least one Product
goto continue_1;  // component with negative mass flow

// ESTIMATE ACTUAL VALUES OF PRODUCT STREAM
PCompPtr = stream_dummy->StCompList;
for(i=0; i<NumOfComps; i++)
{
    PCompPtr->CompMassFlow = mFlowOut[i];
PCompPtr = PCompPtr->next;
}

// ESTIMATE PROPERTIES OF PRODUCT STREAM
Copy_Stream( stream_dummy, strProd );
strProd->Temp = ntrPtr->ReacTemp;

// ESTIMATE POWER REQUIREMENT
AgiPower = AgitRate * LiqVol;
if( !unitPtr->m_isPowerSet )
    unitPtr->Power = AgiPower;

// ESTIMATE HEAT DUTY - BASED ON REFERENCE COMPONENT CONSUMPTION
strcpy( RxnCompName, ntrPtr->RxnHeatComp );
RxnCompPtrIn1 = Find_CompPtr( strFeed, RxnCompName );
RxnCompPtrIn2 = Find_CompPtr( strNeutral, RxnCompName );
RxnCompPtrOut = Find_CompPtr( strProd, RxnCompName ) ;
if( RxnCompPtrIn1 != NULL && RxnCompPtrIn2 != NULL && RxnCompPtrOut != NULL )
{
    RxnCompIn = RxnCompPtrIn1->CompMassFlow + RxnCompPtrIn2->CompMassFlow;
    RxnCompOut = RxnCompPtrOut->CompMassFlow;
    RxnCompMFlow = RxnCompIn - RxnCompOut;
    Duty = ntrPtr->RxnHeat * fabs(RxnCompMFlow);
}
else
    Duty = 0.0;

Mix2_Stream( strFeed, strNeutral, stream_dummy );
Duty += Stream_Enthalpy(strProd) - Stream_Enthalpy(stream_dummy);
Duty -= AgiPower * (3600.0/4.18); // kW -> kcal/h
if( Duty >= 0.000001 && !unitPtr->m_isHeatingDutySet)
    unitPtr->Duty[0] = Duty;
else if( Duty < -0.0001 && !unitPtr->m_isCoolingDutySet )
    unitPtr->Duty[1] = Duty;
if( Duty >= 0.0 )
    unitPtr->DutyTemp[0] = ntrPtr->ReacTemp;
else
    unitPtr->DutyTemp[1] = ntrPtr->ReacTemp;

// ESTIMATE COST DATA
Cost_Neutralizer( unitPtr );
// ADJUST UTILITIES BY CONSIDERING THE NUMBER OF UNITS
Number = unitPtr->Number;

if( !unitPtr->m_isPowerSet )
unitPtr->Power /= Number;

if( !unitPtr->m_isHeatingDutySet )
unitPtr->Duty[0] /= Number;
if( !unitPtr->m_isCoolingDutySet )
unitPtr->Duty[1] /= Number;

// ESTIMATE ENVIRONMENTAL PROPERTIES OF EXITING STREAMS
Estimate_EnviroStreamProperties(strProd);

/******************************
* Solve_AerationBasin(): estimates the size and purchase cost of an Aeration Basin *
******************************/
Solve_AerationBasin(LINK unitPtr)
{
AerationBasinPtr arbPtr;
St_LINK FeedStr, AirInStr;
St_LINK ProdStr, AirOutStr;
St_LINK shead;
St_LINK assStream[2];
St_CompPtr FeedCompPtr, ProdCompPtr;
St_CompPtr AirInCompPtr, AirOutCompPtr;
St_CompPtr BiomassCompPtr, H2OCompPtr, CO2CompPtr, OxygenCompPtr;
short i, j;
short StrNum;

double FeedMassFlow; // kg/h
double VolRemovFlow; // Amount removed due to stripping, kg/h
double SorRemovFlow; // Amount removed due to sorption, kg/h
double BioRemovFlow; // Amount removed due to biodegradation
double TotalBiodegradation;
double IntraIn, IntraOut; // Intracellular amount in inlet-exit
double ExtraIn, ExtraOut; // Extracellular amount in inlet-exit
double LiqVol, TotVol; // m3
double ResTime; // h
double VolRatio, HDRatio, factor;
double Q, theta, Yield, DO;
double TotalOxygenRequired;
double RequiredOxygenRate;
double OxygenFlow;
double FeedVolFlow, BiomassFlow;
double Duty, Power;
double NewIntra, NewExtra;
short StreamPoints;

arbPtr = (AerationBasinPtr) unitPtr->theModel;
/ FIND FEED STREAMS
shhead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shhead, FEED, &StrNum );

StreamPoints = assStream[0]->NumOfPoints;
if( EqualPt( assStream[0]->spoints[StreamPoints-1], unitPtr->hpoints[1] ) )
{
    AirInStr = assStream[0];
    FeedStr = assStream[1];
}
else
{
    AirInStr = assStream[1];
    FeedStr = assStream[0];
}

// FIND PRODUCT STREAMS
shhead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shhead, PROD, &StrNum );

if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] ) )
{
    ProdStr = assStream[0];
    AirOutStr = assStream[1];
}
else
{
    ProdStr = assStream[1];
    AirOutStr = assStream[0];
}

// CHECK IF FEED FLOWRATE IS ZERO
FeedMassFlow = FeedStr->MassFlow;
if( FeedMassFlow < 0.0000000001 )
    return;

// ESTIMATE VOLUMETRIC FLOWRATE
Q = Estim_VolFlow(FeedStr);

// ESTIMATE BOD5 LOADING IN THE FEED STREAM FOR COSTING PURPOSES, kg/d
BODin = FeedStr->BOD5_Stream;
unitPtr->Size = (BODin*Q)*(24.0/1000.0);

// GET CURRENT VALUES OF THE DATA STRUCTURE
Yield = arbPtr->YieldCoeff;
DO = arbPtr->DO;

// CHECK IF HYDRAULIC RESIDENCE TIME IS SPECIFIED
if(!arbPtr->LiqVolSpec)
{
    theta = arbPtr->HydResTime;
    arbPtr->LiqVol = theta*Q;
}
else
{
    theta = arbPtr->LiqVol/Q;
    arbPtr->HydResTime = theta;
224

}/**
 // ESTIMATE TOTAL BASIN VOLUME, m3
 arbPtr->BasinVolume = arbPtr->LiqVol/arbPtr->VolRatio;

 // FIND OXYGEN REQUIREMENT, kg
 TotalOxygenRequired = DO*arbPtr->LiqVol/1000.0;

 // FIND REQUIRED OXYGEN RATE, kg/h
 RequiredOxygenRate = TotalOxygenRequired/theta;

 // ADJUST AIR STREAM FLOWRATE
 OxygenCompPtr = Find_CompPtr(AirInStr, oxygenComp );
 if( OxygenCompPtr == NULL )
 {  
     BioAlert(BIO_ALRT_OK, "The Oxygen Component has not been specified. Please specify it through Component Initialization because simulation of Aeration Basin cannot proceed without it.");
     return;  
 }
 OxygenFlow = OxygenCompPtr->CompMassFlow;
 if(OxygenFlow > 1.0e-16)
     factor = RequiredOxygenRate / OxygenFlow;
 else
 {  
     BioAlert(BIO_ALRT_OK, "The flow of the Oxygen Component in the gas feed is zero. Please initialize its value.");
     return;  
 }
 Multiply_Stream(AirInStr, factor);
 Mix2_Stream( FeedStr, AirInStr, stream dummy);
 Estimate_EnviroStreamProperties(stream dummy);

 // ESTIMATE BIOMASS IN FEED
 BiomassCompPtr = Find_CompPtr(stream dummy, biomassComp );
 if( BiomassCompPtr == NULL )
 {  
     BioAlert(BIO_ALRT_OK, "The Biomass Component has not been specified. Please specify it through Component Initialization because simulation of Aeration Basin cannot proceed without it.");
     return;  
 }
 FeedVolFlow = Estim_VolFlow(FeedStr);
 BiomassFlow = BiomassCompPtr->CompMassFlow;

 // MASS BALANCES
 Copy_Stream(stream dummy, ProdStr);
 FeedCompPtr = stream dummy->StCompList;
ProdCompPtr = ProdStr->StCompList;
AirOutCompPtr = AirOutStr->StCompList;

// CALCULATION OF THE EXTENTS OF BIODEGRADATION, SORPTION, AND STRIPPING
VolRemovFlow = 0.0; // REMOVAL DUE TO STRIPPING
SorRemovFlow = 0.0; // REMOVAL DUE TO SORPTION
BioRemovFlow = 0.0; // REMOVAL DUE TO BIODEGRADATION
TotalBiodegradation = 0.0; // TOTAL AMOUNT OF COMPOUNDS THAT ARE BIODEGRADE

for( i=0; i<NumOfComps; i++)
{
    SorRemovFlow = FeedCompPtr->CompMassFlow * FeedCompPtr->CompLocation[1] * arbPtr->Ext_of_Sor[i];
    BioRemovFlow = FeedCompPtr->CompMassFlow * arbPtr->Ext_of_Bio[i];
    VolRemovFlow = FeedCompPtr->CompMassFlow * arbPtr->Ext_of_Stp[i];

    TotalBiodegradation += BioRemovFlow;

    if( SorRemovFlow > 1.0e-16 )
    {
        IntraIn = FeedCompPtr->CompMassFlow * FeedCompPtr->CompLocation[0];
        ExtraIn = FeedCompPtr->CompMassFlow - IntraIn;

        IntraOut = IntraIn + SorRemovFlow;
        ExtraOut = ExtraIn - SorRemovFlow;

        if( (IntraOut + ExtraOut) > 1.0e-16 )
        {
            ProdCompPtr->CompLocation[0] = IntraOut/(IntraOut+ExtraOut);
            ProdCompPtr->CompLocation[1] = 1.0 - ProdCompPtr->CompLocation[0];
        }
        else
        {
            ProdCompPtr->CompLocation[0] = 0.0;
            ProdCompPtr->CompLocation[1] = 1.0;
        }
    }

    ProdCompPtr->CompMassFlow -= (BioRemovFlow + VolRemovFlow);
    AirOutCompPtr->CompMassFlow = VolRemovFlow;

    FeedCompPtr = FeedCompPtr->next;
    ProdCompPtr = ProdCompPtr->next;
    AirOutCompPtr = AirOutCompPtr->next;
}

BiomassCompPtr = Find_CompPtr(ProdStr, biomassComp);
if( BiomassCompPtr == NULL )
{
    BioAlert(BIO_ALRT_OK, "The Biomass Component has not been specified. Please specify it through Component Initialization because simulation of Aeration Basin cannot proceed without it.");
    return;
}
H2OCompPtr = Find_CompPtr(ProdStr, waterComp);
if( H2OCompPtr == NULL )
{
    BioAlert(BIO_ALRT_OK, "The Water Component has not been specified. Please
specify it through Component Initialization because simulation of
Aeration Basin cannot proceed without it.");
    return;
}

CO2CompPtr = Find_CompPtr(AirOutStr, arbPtr->m_carbonDioxideComp );
if( CO2CompPtr == NULL )
{
    BioAlert(BIO_ALRT_OK, "The Carbon Dioxide Component has not been
specified. Please specify it because simulation of Aeration Basin
cannot proceed without it.");
    return;
}

BiomassCompPtr->CompMassFlow += Yield * TotalBiodegradation;
H2OCompPtr->CompMassFlow += 0.5 * (1.0 - Yield) * TotalBiodegradation;
CO2CompPtr->CompMassFlow += 0.5 * (1.0 - Yield) * TotalBiodegradation;

if(CO2CompPtr->CompMassFlow < 0.0000001)
    CO2CompPtr->CompMassFlow = 0.0;

H2OCompPtr->CompMassFlow += TotalPhotodegradation;
ProdStr->MassFlow = Estim_MassFlow(ProdStr);
AirOutStr->MassFlow = Estim_MassFlow(AirOutStr);

ProdStr->MassFlow = Estim_MassFlow( ProdStr );
AirOutStr->MassFlow = Estim_MassFlow( AirOutStr );

// ESTIMATE ELECTRIC POWER REQUIREMENT
Power = arbPtr->PowerInput * arbPtr->LiqVol;
if( !unitPtr->m_isPowerSet )
    unitPtr->Power = Power;

// ESTIMATE COST DATA
Cost_AerationBasin( unitPtr );

// ESTIMATE ENVIRONMENTAL PROPERTIES OF EXITING STREAMS
Estimate_EnviroStreamProperties(ProdStr);
Estimate_EnviroStreamProperties(AirOutStr);

/**************************************************************************
* SOLVE TRICKLING FILTER
* 
* Programmed by: Kostas Abeliotis
* Date: 5/31/1993
* 
* Reference 1: Wastewater Engineering, Metcalf & Eddy Inc.
* Paragraphs 10.5 and 8.8 pp 614,403
* 
* Reference 2: Design of Municipal Wastewater Treatment Plants
* Volume 1, Chapter 12, pp 677
* */
Solve_TricklingFilter(LINK unitPtr)
{
    TricklingFilterPtr trfPtr;
    St_Compptr ProductCompPtr, FeedCompPtr, AirOutCompPtr, BiomassCompPtr;
    St_LINK FeedStr, ProdStr, AirOutStr, shead;
    St_LINK assStream[2];
    short i, StrNum;
    short StreamPoints;

    double mflow; // Feed mass flowrate, kg/h
    double HydLoad; // Filter Hydraulic Loading, m3/m2-h
    double Area; // Filter Cross Sectional Area, m2
    double Depth; // Filter Depth, m
    double Rf; // Recycle Factor
    double alpha; // Recycle Ratio
    double delta; // Weighing Factor
    double n; // Exponential term
    double Ko; // Treatability Factor
    double beta; // dummy variable
    double theta; // Temp. correction of treatability factor
    double OperTemp; // Operating Temperature, °C
    double FilterVolume; // Total filter volume, m3
    double Q; // Volumetric flowrate, m3/h
    double Si; // Inlet BOD5, mg/l
    double Se; // Exit BOD5, mg/l
    double Xout; // Exit Biomass concentration, mg/l
    double OverallEff; // Overall BOD5 Removal efficiency
    double term; // dummy variable
    double Yield; // Biomass yield on substrate removed, g/g
    double Eff[compMax]; // Biodegradation component fraction
    double Stp[compMax]; // Stripping component fraction
    double AmountBiodegraded; // Amount of component biodegraded
    double TotalAmountBiodegraded; // Total Amount biodegraded
    double FeedBioDegradable; // Flow of biodegradable components, kg/h

    trfPtr = (TricklingFilterPtr) unitPtr->theModel;

    // FIND FEED STREAM
    shead = stream_head;
    Find_Ass_Stream(unitPtr, assStream, shead, FEED, &StrNum);
    FeedStr = assStream[0];

    // FIND PRODUCT STREAMS
    shead = stream_head;
    Find_Ass_Stream(unitPtr, assStream, shead, PROD, &StrNum);
    if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] ) )
    {
        ProdStr = assStream[1];
        AirOutStr = assStream[0];
    }
    else
    {
        ProdStr = assStream[0];
        AirOutStr = assStream[1];
    }
// CHECK IF FEED MASS FLOWRATE IS ZERO
mflow = FeedStr->MassFlow;
if( mflow < 0.0000001 )
    return;

// ESTIMATE FEED VOLUMETRIC FLOWRATE, m3/h
Q = Estim_NonGaseousVolFlow(FeedStr);

// GET CURRENT VALUES OF DATA STRUCTURE
HydLoad = trfPtr->HydLoading;
Depth = trfPtr->Depth;
alpha = trfPtr->RecycleRatio;
delta = trfPtr->WeighingFactor;
n = trfPtr->n_constant;
Ko = trfPtr->Treat_Factor;
theta = trfPtr->theta;
OperTemp = trfPtr->OperTemp;
Yield = trfPtr->Yield;

for(i=0; i<compMax; i++)
{
    Eff[i] = trfPtr->Ext_of_Bio[i];
    Stp[i] = trfPtr->Ext_of_Stp[i];
}

// CALCULATE CORRECTED VALUE OF TREATABILITY FACTOR
Ko = Ko*pow(theta,(OperTemp - 20.0));

// FIND FILTER CROSS SECTIONAL AREA, m2
Area = (1.0 + alpha)*Q/HydLoad;

// FIND ENTERING BODS CONCENTRATION, mg/l
Si = FeedStr->BOD5_Stream;

// ESTIMATE ORGANIC LOADING, kg BODS/d
trfPtr->OrgLoading = (Si*Q)*(24.0/1000.0);

// FIND FLOWRATE OF COMPONENTS THAT ARE BIODEGRADED, kg/h
FeedCompPtr = FeedStr->StCompList;
FeedBioDegradable = 0.0;
for( i=0; i<NumOfComps; i++)
{
    if(trfPtr->Ext_of_Bio[i] > 0.0)
        FeedBioDegradable += FeedCompPtr->CompMassFlow;
    FeedCompPtr = FeedCompPtr->next;
}

// MASS BALANCES
Copy_Stream(FeedStr, ProdStr);

FeedCompPtr = FeedStr->StCompList;
ProductCompPtr = ProdStr->StCompList;
AirOutCompPtr = AirOutStr->StCompList;

TotalAmountBiodegraded = 0.0;
for( i=0; i<NumOfComps; i++)
{
    AmountBiodegraded = ProductCompPtr->CompMassFlow*Eff[i];
    TotalAmountBiodegraded += AmountBiodegraded;

    ProductCompPtr->CompMassFlow = FeedCompPtr->CompMassFlow*(1.0 - Eff[i]);
    AirOutCompPtr->CompMassFlow = FeedCompPtr->CompMassFlow*Stp[i];

    FeedCompPtr = FeedCompPtr->next;
    ProductCompPtr = ProductCompPtr->next;
    AirOutCompPtr = AirOutCompPtr->next;
}

ProdStr->MassFlow = Estim_MassFlow( ProdStr );
AirOutStr->MassFlow = Estim_MassFlow( AirOutStr );

// CALCULATE OVERALL EFFICIENCY
OverallEff = TotalAmountBiodegraded/FeedBioDegradable;

// ACCOUNT FOR YIELD OF BIOMASS ON SUBSTRATE
BiomassCompPtr = Find_CompPtr(ProdStr, "Biomass");
BiomassCompPtr->CompMassFlow += Yield * TotalAmountBiodegraded;

ProdStr->MassFlow = Estim_MassFlow( ProdStr );
AirOutStr->MassFlow = Estim_MassFlow( AirOutStr );

// SIZING CALCULATIONS FOR NRC MODEL
if(trfPtr->NRC_Model)
{
    // CALCULATE RECYCLE FACTOR
    Rf = (1.0 + alpha)/pow(1.0 + (1.0 - delta)*alpha,2.0);
    term = OverallEff/(1.0 - OverallEff);
    FilterVolume = (0.000196*(Q*24.0)*Si/Rf)*pow(term,2.0);
    Depth = FilterVolume/Area;
}
else // ECKENFELDER MODEL
{
    // FIND EXITING BOD5 CONCENTRATION, mg/l
    Se = (1.0 - OverallEff)*Si;
    term = Se*(1.0 + alpha)/(Si + alpha*Se);
    Depth = - log(term)/(Ko*pow(1.0/((1.0 + alpha)*Q),n));
}

// ASSIGN NEW VALUES TO DATA STRUCTURE
trfPtr->Depth = Depth;
trfPtr->Area = Area;

// ESTIMATE COST
Cost_TricklingFilter( unitPtr);

// ESTIMATE ENVIRONMENTAL PROPERTIES OF EXITING STREAMS
Estimate_EnviroStreamProperties(ProdStr);
Estimate_EnviroStreamProperties(AirOutStr);

/she **SOLVE THICKENER**

Solve_Thickener(LINK unitPtr)
{
ThickenerPtr thkPtr;
St_CompPtr ClarifiedCompPtr, SludgeCompPtr, FeedCompPtr;
St_LINK FeedStr, SludgeStr;
St_LINK ClarifiedStr, shead;
St_LINK assStream[2];
short i, StreamNum;
short StreamPoints;

short ModelType; // Type of Model to be used for settling flux estimation
double SetFlux; // Settling Velocity, m3/m2-h
double SVI; // Sludge volume index, mL/g
double Param1; // Parameter in Generic Model, m/h
double Param2; // Parameter in Generic Model, mL/g
double Area; // Thickener area, m2
double Diam; // Diameter of circular Thickener, m
double Length; // Length of rectangular Thickener, m
double Width; // Width of rectangular Thickener, m
double L_W_Ratio; // Ratio Length/Width for rectangular Clarifier
double Part_Conc_In_Sludge; // Fraction of solids in sludge, vol/vol,
double RemovFrac[compMax]; // Removal fraction for each particle component
double mflow; // Entering mass flow rate, kg/h
double Q; // Entering volumetric flow rate, m3/h
double AmountRemoved; // Amount removed from each component
double Part_Vol_In_Sludge; // Vol. flow of solids in the sludge stream, m3/
double Tot_Sol_In_Sludge; // Flow of solids in the sludge stream, kg/h
double TotVol_of_Sludge; // Total Vol. flow of the sludge stream, m3/h
double FeedLiqVol; // Vol. flow of liquid in the feed stream, m3/h
double Frac_of_FeedLiqVol_In_Sludge; // Fraction of the feed liquid volume
  // that goes to the sludge stream
double solventInSludge; // solvent component going into the sludge stream

double Tot_Sol_In_Feed; // Flow of solids in the feed stream, [kg/h]
double FeedSolConc; // Solid Concentration in feed stream, kg/m3

short biomassIndex;
double intraInSludge, extraInSludge, totalInSludge;
double intraRemovalFrac, massFlowIn;

double intraInSludgeBefore, extraInSludgeBefore;
double intraInSludgeAfter, extraInSludgeAfter;

double totalInSludgeAfter;

thkPtr = (ThickenerPtr) unitPtr->theModel;

/* FIND FEED AND PRODUCT STREAMS */
shedd = stream_head;
Find_Ass_Stream( unitPtr, assStream, shedd, FEED, &StrNum );
FeedStr = assStream[0];

shedd = stream_head;
Find_Ass_Stream( unitPtr, assStream, shedd, PROD, &StrNum );
if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] )
{  
    ClarifiedStr = assStream[0];
    SludgeStr = assStream[1];
}
else
{  
    ClarifiedStr = assStream[1];
    SludgeStr = assStream[0];
}

// GET CURRENT DATA STRUCTURE VALUES
SVI = thkPtr->SVI;  // mL/g
Part_Conc_In_Sludge = (thkPtr->Part_Conc_In_Sludge)*1.0e-3;  // mg/lt to kg/m3

mflow = FeedStr->MassFlow;

// CHECK IF FEED FLOWRATE IS ZERO
if( mflow < 0.0000001 )
{
    BioAlert(BIO_ALRT_OK, "The mass flowrate of the feed stream to the 'Thickener' is zero. Please increase it so that the program can function properly ");
    return;
}

// ESTIMATE FEED STREAM SOLID CONCENTRATION , kg/m3
Tot_Sol_In_Feed = Estim_SolidMassFlow( FeedStr );
FeedSolConc = Tot_Sol_In_Feed/Estim_VolFlow(FeedStr);

// CHECK IF THE SOLIDS CONCENTRATION IN FEED STREAM IS GREATER THAN THE DESIRED CONCENTRATION IN SLUDGE STREAM
if( (FeedSolConc - Part_Conc_In_Sludge) > 1.0e-8)
{
    BioAlert(BIO_ALRT_OK, "The concentration of solids in the feed stream is greater than the desired concentration of solids in the thickened sludge stream. Please check your feed stream data or increase the solids concentration in sludge ");
    return;
}

// CHECK IF USER SPECIFIES SETTLING VELOCITY OR NOT
if(thkPtr->SetFluxSpec)
    SetFlux = thkPtr->SetFlux,
else  // Model estimates settling velocity
{

  // CALCULATE SETTLING VELOCITY BASED ON MODEL PARAMETERS, m/h
  if(thkPtr->ModelType == 1)  // Wahlberg and Keinath Model
  {
    // ESTIMATION OF MODEL PARAMETERS
    Param1 = 15.3 - 0.0615*SVI;
    Param2 = 0.426 - 0.00384*SVI + 0.0000543*pow(SVI,2.0);
  }
  else if(thkPtr->ModelType == 2)  // Daigger and Roper Model
  {
    // ESTIMATION OF MODEL PARAMETERS
    Param1 = 7.8;
    Param2 = 0.148 + 0.0021*SVI;
  }
  else  // Generic Model
  {
    // ESTIMATION OF MODEL PARAMETERS
    Param1 = thkPtr->Param1;
    Param2 = thkPcr->Param2 / 1000.0;  // Convert mL/g to L/g
  }

  SetFlux = Param1 * exp(- Param2 * FeedSolConc);
  thkPtr->SetFlux = SetFlux;
 }

  // ESTIMATE ENTERING VOLUMETRIC FLOW RATE [m3/h]
  Q = Estim_VolFlow( FeedStr );

  // CALCULATE THE SURFACE AREA OF THE THICKENER, m2
  Area = Q / SetFlux;
  thkPtr->Area = Area;

  // CHECK IF CIRCULAR DESIGN IS USED
  if(thkPtr->Circular)  // Circular Design
  {
    Diam = pow((4.0 * Area / PI),0.5);
    thkPtr->Diam = Diam;

    thkPtr->Width = 0.0;  // Set rectangular characteristics to zero
    thkPtr->Length = 0.0;
  }
  else  // Rectangular design
  {
    L_W_Ratio = thkPtr->L_W_Ratio;
    Width = pow((Area/L_W_Ratio),0.5);
    Length = L_W_Ratio * Width;
    thkPtr->Width = Width;
    thkPtr->Length = Length;

    thkPtr->Diam = 0.0;  // Set circular characteristics to zero
  }

  // GET INDEX OF BIOMASS COMP AND REMOVAL FRAC OF INTRACELLULAR COMPS
  intraRemovalFrac = 0.0;
  biomassIndex = GetComponentIndex( biomassComp );
if( biomassIndex != -1 )
        intraRemovalFrac = thkPtr->RemovFrac[biomassIndex];

    // CONSIDER REMOVAL OF PARTICULATE AND INTRACELLULAR COMPONENTS
    FeedCompPtr = FeedStr->StCompList;
    SludgeCompPtr = SludgeStr->StCompList;

    for( i=0; i<NumOfComps; i++ )
    {
        intraRemovalFrac = MAX( intraRemovalFrac, thkPtr->RemovFrac[i] );

        massFlowIn = FeedCompPtr->CompMassFlow;
        intraInSludge = massFlowIn * FeedCompPtr->CompLocation[0] * 
                        intraRemovalFrac;
        extraInSludge = massFlowIn * FeedCompPtr->CompLocation[1] * 
                        thkPtr->RemovFrac[i];
        totalInSludge = intraInSludge + extraInSludge;

        SludgeCompPtr->CompMassFlow = totalInSludge;
        if( totalInSludge > 0.000001 )
        {
            SludgeCompPtr->CompLocation[0] = intraInSludge/totalInSludge;
            SludgeCompPtr->CompLocation[1] = extraInSludge/totalInSludge;
        }
        else
        {
            SludgeCompPtr->CompLocation[0] = 0.0;
            SludgeCompPtr->CompLocation[1] = 1.0;
        }

        FeedCompPtr = FeedCompPtr->next;
        SludgeCompPtr = SludgeCompPtr->next;
    }

    SludgeStr->MassFlow = Estim_MassFlow( SludgeStr );

    // FLOW RATE OF SOLIDS IN SLUDGE STREAM, kg/h
    Tot_Sol_In_Sludge = Estim_SolidMassFlow( SludgeStr );

    // VOLUMETRIC FLOW RATE OF SOLIDS IN SLUDGE STREAM, m3/h
    Part_Vol_In_Sludge = Estim_VolFlow(SludgeStr);

    // TOTAL VOLUMETRIC FLOW RATE OF SLUDGE STREAM, m3/h
    TotVol_of_Sludge = Tot_Sol_In_Sludge / Part_Conc_In_Sludge;

    // VOLUMETRIC FLOW RATE OF LIQUID IN FEED STREAM, m3/h
    FeedLiqVol = Estim_SolventVolFlow(FeedStr);

    // FRACTION OF THE FEED LIQUID THAT GOES TO THE SLUDGE STREAM
    Frac_of_FeedLiqVol_In_Sludge = (TotVol_of_Sludge - Part_Vol_In_Sludge) / 
                                FeedLiqVol;

    if( Frac_of_FeedLiqVol_In_Sludge > 1.0 )
        Frac_of_FeedLiqVol_In_Sludge = 1.0;
    if( Frac_of_FeedLiqVol_In_Sludge < 0.0 )
Frac_of_FeedLiqVol_In_Sludge = 0.0;

// MASS BALANCES FOR THE EXTRACELLULAR FRACTION OF LIQUID COMPONENTS

FeedCompPtr = FeedStr->StCompList;
SludgeCompPtr = SludgeStr->StCompList;

for( i=0; i<NumOfComps; i++)
{
    if( thkPtr->RemovFrac[i] < 0.0000001)
    {
        intraInSludgeBefore = SludgeCompPtr->CompMassFlow * SludgeCompPtr->CompLocation[0];
        extraInSludgeBefore = SludgeCompPtr->CompMassFlow * SludgeCompPtr->CompLocation[1];

        solventInSludge = FeedCompPtr->CompMassFlow * FeedCompPtr->CompLocation[1] * Frac_of_FeedLiqVol_In_Sludge;

        intraInSludgeAfter = intraInSludgeBefore;
        extraInSludgeAfter = extraInSludgeBefore + solventInSludge;
        totalInSludgeAfter = intraInSludgeAfter + extraInSludgeAfter;

        SludgeCompPtr->CompMassFlow = totalInSludgeAfter;
        if( totalInSludgeAfter > 0.000001 )
        {
            SludgeCompPtr->CompLocation[0] = intraInSludgeAfter/totalInSludgeAfter;
            SludgeCompPtr->CompLocation[1] = extraInSludgeAfter/totalInSludgeAfter;
        }
        else
        {
            SludgeCompPtr->CompLocation[0] = 0.0;
            SludgeCompPtr->CompLocation[1] = 1.0;
        }
    }

    FeedCompPtr = FeedCompPtr->next;
    SludgeCompPtr = SludgeCompPtr->next;
}

SludgeStr->MassFlow = Estim_MassFlow( SludgeStr );

// CALCULATE FLOWRATE AND COMPOSITION OF CLARIFIED STREAM
Subtract_Stream( FeedStr, SludgeStr, ClarifiedStr );

/* ESTIMATE PURCHASE COST OF THICKENER */
Cost_Thickener( unitPtr );

/* ESTIMATE ENVIRONMENTAL PROPERTIES OF EXITING STREAMS */
Estimate_EnviroStreamProperties(ClarifiedStr);
Estimate_EnviroStreamProperties(SludgeStr);
Solve_AnaerobicDigestion(): estimates the size and purchase cost of the Anaerobic Digestion Process

Solve_AnaerobicDigestion(LINK unitPtr)
{
    AnaerobicDigestionPtr andPtr;
    St_LINK FeedStr;
    St_LINK ProdStr, AirOutStr, SludgeStr;
    St_LINK shead;
    St_LINK assStream[4];
    St_CompPtr FeedCompPtr, ProdCompPtr;
    short i, j;
    short StrNum;

double FeedMassFlow; // Feed mass flowrate, kg/d
    double VolRemovFlow; // Amount going to the gas outlet stream
    double SludgeRemovFlow; // Amount going to the sludge stream
    double IntraIn, IntraOut; // Intra cellular amount in inlet-exit
    double ExtraIn, ExtraOut; // Extra cellular amount in inlet-exit
    double Q; // Feed volumetric flowrate, m³/d
    double DigestTemp; // Digest. Temperature °C for high rate digester
    double PowerInput; // KW/m³
    double HeatEff; // Heating Units Efficiency
    double HeatingValue; // Heating Value of digester gas, kcal/m³
    double SolidConc; // Solid Concentration in sludge stream, kg/m³
    double GasProduct; // Gas Production, m³/h
    double DigestGasRequired; // Digestion Gas Required for heating, m³/h
    double AddHeatDuty; // Additional Heating Duty, kcal/h
    double NaturalGasRequired; // Gas Required for additional heating, m³/h
    double HeatDuty; // Heating duty Required for heating the feed stream from feed Temp to Digest temp, kcal/h
    double HeatDutyProduced; // Duty of the produced Digestion gas, kcal/h
    double DH_Feed; // Enthalpy of feed stream @ Feed Temp, kcal/kg
    double DH_Digest; // Enthalpy of feed stream @ digester T, kcal/kg
    double intraRemovalFrac; // Remov. Frac. of intracellular comp.
    double intraInSludge; // Amount of intracellular in sludge, kg/h
    double extraInSludge; // Amount of extra-cellular in sludge, kg/h
    double Tot_Sol_In_Feed; // Flow of solids in the feed stream, [kg/h]
    double Tot_Sol_In_Sludge; // Flow of solids in the sludge stream, [kg/h]
    double Part_Vol_In_Sludge; // Vol. flow of solids in sludge stream, [m³/h]
    double Tot_Vol_of_Sludge; // Vol. flow of the sludge stream, [m³/h]
    double FeedLiqVol; // Vol. flow of liquid in the feed stream, [m³/h]
    double frac_of_FeedLiqVol_In_Sludge; // Fraction of the feed liquid volume that goes to the sludge stream
    double FeedSolConc; // Solid Concentration in feed stream, kg/m³
    double intraInSludgeBefore, extraInSludgeBefore;
    double intraInSludgeAfter, extraInSludgeAfter;
    double solventInSludge, totalInSludgeAfter;
    double Extent, pivot;
double mFlowOut[compMax];
double DeltaMass;
double mFlowKey, mFlowIn;
double Ai,Ak,Ratio_Max,Ratio;
double Intra, Extra;
short index;

double Power;

andPtr = (AnaerobicDigestionPtr) unitPtr->theModel;

// FIND FEED STREAM
shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );
FeedStr = assStream[0];

// FIND PRODUCT STREAMS
shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );

for(i=0; i<StrNum; i++)
{
    if( EqualPt( assStream[i]->spoints[0], unitPtr->hpoints[3] ) )
    {
        AirOutStr = assStream[i];
        break;
    }
}

for(i=0; i<StrNum; i++)
{
    if( EqualPt( assStream[i]->spoints[0], unitPtr->hpoints[2] ) )
    {
        ProdStr = assStream[i];
        break;
    }
}

for(i=0; i<StrNum; i++)
{
    if( EqualPt( assStream[i]->spoints[0], unitPtr->hpoints[1] ) )
    {
        SludgeStr = assStream[i];
        break;
    }
}

// CHECK IF FEED FLOWRATE IS ZERO
FeedMassFlow = FeedStr->MassFlow;
if( FeedMassFlow < 1.0e-8 )
{
    BioAlert(BIO_ALRT_OK, "The mass flowrate of the feed stream to the 'Aerobic Digester' is zero. Please increase it so that the program can function properly!");
    return;
}
// ESTIMATE FEED FLOW TO THE DIGESTER FOR COSTING PURPOSES, m³/h
unitPtr->Size = Estim_VolFlow(FeedStr);

// ESTIMATE (SLUDGE) FEED LOADING TO THE DIGESTER FOR COSTING PURPOSES, kg/d
// unitPtr->Size = FeedMassFlow*24.0;

// ESTIMATE FEED VOLUMETRIC FLOWRATE, m³/d
Q = Estim_VolFlow(FeedStr)*24.0;

// GET CURRENT VALUES OF THE DATA STRUCTURE
DigestTemp = andPtr->DigestTemp;
Powerinput = andPtr->PowerInput;
Extent = andPtr->UtilEff;
HeatEff = andPtr->HeatEff;
HeatingValue = andPtr->HeatingValue;
SolidConc = andPtr->SolidConc * 1.0e-3; // Convert mg/lt to kg/m³

// CHECK IF SOLIDS RETENTION TIME (DAYS) IS SPECIFIED
if(!andPtr->VolumeSpec)
  andPtr->ActiveVol = andPtr->SRT*Q;
else
  andPtr->SRT = andPtr->ActiveVol/Q;

// ESTIMATE TOTAL DIGESTER VOLUME, m³
andPtr->TotalVol = andPtr->ActiveVol/andPtr->Ratio;

// MASS BALANCES
Copy_Stream(FeedStr,ProdStr);
ProdCompPtr = ProdStr->StCompList;

Ratio_Max = -32000000.0;
for(i=0; i<NumOfComps; i++)
{
  if( andPtr->Stoich[i] < 0.0 )
  {
    Ratio = ProdCompPtr->CompMassFlow/andPtr->Stoich[i];
    if(Ratio > Ratio_Max)
      Ratio_Max = Ratio;
  }
  ProdCompPtr = ProdCompPtr->next;
}
ProdCompPtr = ProdStr->StCompList;

for(i=0; i<NumOfComps; i++)
{
  if( andPtr->Stoich[i] < -0.0000001 )
  {
    mFlowIn = ProdCompPtr->CompMassFlow;
    if( mFlowIn/andPtr->Stoich[i] >= Ratio_Max )
    {
      Ratio_Max = mFlowIn/andPtr->Stoich[i];
      Ak = andPtr->Stoich[i];
      mFlowKey = ProdCompPtr->CompMassFlow;
    }
  }
}
ProdCompPtr = ProdCompPtr->next;
}

// ESTIMATE OUTLET MASS FLOWS BASED ON THE ABOVE KEY COMPONENT
continue_1:
FeedCompPtr = FeedStr->StCompList;
for (i=0; i<NumOfComps; i++)
{
    mFlowIn = FeedCompPtr->ComoMassFlow;
    Ai = andPtr->Stoich[i];
    mFlowOut[i] = mFlowIn - mFlowKey * ( Ai / Ak * Extent );
    FeedCompPtr = FeedCompPtr->next;
}

// CHECK IF ANY OUTLET FLOW IS NEGATIVE AND IF YES FIND THE MINIMUM
pivot = 0.001;
FeedCompPtr = FeedStr->StCompList;
for (i=0; i<NumOfComps; i++)
{
    if (mFlowOut[i] < 0.0 && mFlowOut[i] < pivot)
    {
        pivot = mFlowOut[i];
        mFlowKey = FeedCompPtr->ComoMassFlow;
        Ak = andPtr->Stoich[i];
    }
    FeedCompPtr = FeedCompPtr->next;
}

if (pivot < 0.0) // there is at least one Product
    goto continue_1; // component with negative mass flow

// ESTIMATE ACTUAL VALUES OF PRODUCT STREAM
Copy_Stream(FeedStr, ProdStr);
ProdCompPtr = ProdStr->StCompList;
AirOutCompPtr = AirOutStr->StCompList;
SludgeCompPtr = SludgeStr->StCompList;

// GET INDEX OF BIOMASS COMP AND REMOVAL FRAC OF INTRACELLULAR COMPS
intraRemovalFrac = 0.0;
biomassIndex = GetComponentIndex(biomassComp);
if (biomassIndex != -1)
    intraRemovalFrac = andPtr->SludgeFrac[biomassIndex];

for (i=0; i<NumOfComps; i++)
{
    if (andPtr->Stoich[i] < 0.0) // reactant component
        ProdCompPtr->ComoMassFlow = mFlowOut[i];
    else // product component
    {
        // BEFORE REACTION
        mFlowIn = ProdCompPtr->ComoMassFlow;
        Intra = mFlowIn * ProdCompPtr->CompLocation[0];
        Extra = mFlowIn * ProdCompPtr->CompLocation[1];
        // AFTER REACTION
        DeltaMass = mFlowOut[i] - mFlowIn;
ProdCompPtr->CompMassFlow += DeltaMass;

if( Intra + Extra < 0.0001 )
{
    ProdCompPtr->CompLocation[0] = 0.0;
    ProdCompPtr->CompLocation[1] = 1.0;
}
else
{
    ProdCompPtr->CompLocation[0] = Intra / (Intra + Extra);
    ProdCompPtr->CompLocation[1] = Extra / (Intra + Extra);
}

// ESTIMATION OF EXITING SLUDGE AND GAS STREAMS

// GAS STREAM
if(andPtr->GasComp[i])
    VolRemovFlow = ProdCompPtr->CompMassFlow*1.0;
else
    VolRemovFlow = 0.0;

// SLUDGE STREAM
intraRemovalFrac = MAX( intraRemovalFrac, andPtr->SludgeFrac[i] );
intraInSludge = ProdCompPtr->CompMassFlow *
    ProdCompPtr->CompLocation[0] * intraRemovalFrac;
extraInSludge = ProdCompPtr->CompMassFlow *
    ProdCompPtr->CompLocation[1] * andPtr->SludgeFrac[i];
SludgeRemovFlow = intraInSludge + extraInSludge;

if( SludgeRemovFlow > 0.000001 )
{
    SludgeCompPtr->CompLocation[0] = intraInSludge/SludgeRemovFlow;
    SludgeCompPtr->CompLocation[1] = extraInSludge/SludgeRemovFlow;
}
else
{
    SludgeCompPtr->CompLocation[0] = 0.0;
    SludgeCompPtr->CompLocation[1] = 1.0;
}

ProdCompPtr->CompMassFlow -= VolRemovFlow;
AirOutCompPtr->CompMassFlow = VolRemovFlow;
SludgeCompPtr->CompMassFlow = SludgeRemovFlow;

ProdCompPtr = ProdCompPtr->next;
AirOutCompPtr = AirOutCompPtr->next;
SludgeCompPtr = SludgeCompPtr->next;

ProdStr->MassFlow = Estim_MassFlow( ProdStr );
AirOutStr->MassFlow = Estim_MassFlow( AirOutStr );
SludgeStr->MassFlow = Estim_MassFlow( SludgeStr );

// CHECK IF THE SOLIDS CONCENTRATION IN FEED STREAM IS GREATER THAN THE
// DESIRED CONCENTRATION IN SLUDGE STREAM
Tot_Sol_In_Feed = Estim_SolioMassFlow( FeedStr );
FeedSolConc = Tot_Sol_In_Feed/Estim_VolFlow(FeedStr);

if( (FeedSolConc - SolidConc) > 1.0e-8 )
{
    BioAlert(BIO_ALRT_OK, "The concentration of solids in the feed stream is
greater than the desired concentration of solids in the sludge stream.
Please check your feed stream data or increase the solids concentration
in sludge !!");
    return;
}

// FLOW RATE OF SOLIDS IN SLUDGE STREAM, kg/h
Tot_Sol_In_Sludge = Estim_SolidMassFlow( SludgeStr );

// VOLUMETRIC FLOW RATE OF SOLIDS IN SLUDGE STREAM, m³/h
Part_Vol_In_Sludge = Estim_VolFlow(SludgeStr);

// TOTAL VOLUMETRIC FLOW RATE OF SLUDGE STREAM, m³/h
TotVol_of_Sludge = Tot_Sol_In_Sludge / SolidConc;

// VOLUMETRIC FLOW RATE OF LIQUID IN FEED STREAM, m³/h
FeedLiqVol = Estim_SolventVolFlow(FeedStr);

// FRACTION OF THE FEED LIQUID THAT GOES TO THE SLUDGE STREAM
Frac_of_FeedLiqVol_In_Sludge = (TotVol_of_Sludge - Part_Vol_In_Sludge) /
FeedLiqVol;

if( Frac_of_FeedLiqVol_In_Sludge > 1.0 )
    Frac_of_FeedLiqVol_In_Sludge = 1.0;

if( Frac_of_FeedLiqVol_In_Sludge < 0.0 )
    Frac_of_FeedLiqVol_In_Sludge = 0.0;

// MASS BALANCES FOR THE EXTRACELLULAR FRACTION OF LIQUID COMPONENTS
FeedCompPtr = FeedStr->StCompList;
SludgeCompPtr = SludgeStr->StCompList;

for( i=0; i<NumOfComps; i++)
{
    if( andPtr->SludgeFrac[i] < 0.0000001 )
    {
        intraInSludgeBefore = SludgeCompPtr->CompMassFlow * 
                             SludgeCompPtr->CompLocation[0];
        extraInSludgeBefore = SludgeCompPtr->CompMassFlow * 
                              SludgeCompPtr->CompLocation[1];

        solventInSludge = FeedCompPtr->CompMassFlow * 
                         FeedCompPtr->CompLocation[1] * Frac_of_FeedLiqVol_In_Sludge;

        intraInSludgeAfter = intraInSludgeBefore;
        extraInSludgeAfter = extraInSludgeBefore + solventInSludge;
        totalInSludgeAfter = intraInSludgeAfter + extraInSludgeAfter;

        SludgeCompPtr->CompMassFlow = totalInSludgeAfter;
        if( totalInSludgeAfter > 0.000001 )
        {

    }
SludgeCompPtr->CompLocation[0] = intraInSludgeAfter/totalInSludgeAfter;
SludgeCompPtr->CompLocation[1] = extraInSludgeAfter/totalInSludgeAfter;
else
{
    SludgeCompPtr->CompLocation[0] = 0.0;
    SludgeCompPtr->CompLocation[1] = 1.0;
}

FeedCompPtr = FeedCompPtr->next;
SludgeCompPtr = SludgeCompPtr->next;

SludgeStr->MassFlow = Estim_MassFlow( SludgeStr );

// CALCULATE FLOWRATE AND COMPOSITION OF SUPERNATANT STREAM
Subtract_Stream( ProdStr, SludgeStr, ProdStr );

// ESTIMATE VOLUMETRIC FLOWRATE OF DIGESTION GAS PRODUCED, m3/h
GasProduct = Estim_VolFlow(AirOutStr);
andPtr->GasProduct = GasProduct;

if( andPtr->HighRate )
{
    // IDENTIFY WATER COMPONENT
    waterindex = GetComponentIndex( waterComp );
    if( waterIndex == -1 )
    {
        BioAlert(BIO_ALRT_OK, "The heat capacity of the sludge in the High
Rate Anaerobic Digestion is assumed equal to that of water. Please
specify the water component, first ");
        return;
    }

    // ENTHALPY OF FEED STREAM, kcal/kg
    DH_Feed = PureComponentEnthalpy(FeedStr->Temp, waterIndex);

    // ENTHALPY OF SLUDGE EXITING THE DIGESTER
    DH_Digest = PureComponentEnthalpy(DigestTemp, waterIndex);

    // HEATING DUTY, kcal/kg
    HeatDuty = DH_Digest - DH_Feed;

    // HEATING DUTY, kcal/h
    HeatDuty *= FeedMassFlow;

    // ACCOUNT FOR 20% HEATING LOSSES (BASED ON "BULK" SLUDGE HEATING DUTY)
    HeatDuty *= 1.2;

    // DIGESTION GAS REQUIRED (WHEN BURNED TO PRODUCE THE HEATING DUTY), m3/
    DigestGasRequired = HeatDuty/(HeatingValue*HeatEff);
    andPtr->DigestGasRequired = DigestGasRequired;

    // CHECK IF DIGESTION GAS REQUIRED IS GREATER THAN THE PRODUCED
if (DigestGasRequired > GasProduct)
{
    // ESTIMATE HEATING DUTY OF GAS PRODUCED, kcal/h
    HeatDutyProduced = GasProduct * HeatingValue * HeatEff;

    // ESTIMATE ADDITIONAL HEATING DUTY, kcal/h
    AddHeatDuty = HeatDuty - HeatDutyProduced;

    // NATURAL GAS REQUIRED TO PRODUCE THE ADDITIONAL HEAT DUTY, m3/h
    // 8900.0 kcal/m3 IS THE HEATING VALUE OF NATURAL GAS
    NaturalGasRequired = AddHeatDuty/(8900.0*HeatEff);
}
else
{
    AddHeatDuty = 0.0;
    NaturalGasRequired = 0.0;
}

andPtr->AddHeatDuty = AddHeatDuty;
andPtr->NaturalGasRequired = NaturalGasRequired;

// ESTIMATE ELECTRIC POWER REQUIREMENT
Power = andPtr->PowerInput * andPtr->ActiveVol;
}
else // STANDARD RATE DIGESTION
{
    HeatDuty = 0.0;
    Power = 0.0;
    DigestTemp = FeedStr->Temp;
}

// ASSIGN HEATING DUTY AND POWER TO UNIT POINTERS
unitPtr->Duty[0] = HeatDuty;
if( !unitPtr->m_isPowerSet )
    unitPtr->Power = Power;

// ASSIGN DIGESTION TEMPERATURE TO EXITING STREAMS
ProdStr->Temp = DigestTemp;
AirOutStr->Temp = DigestTemp;
SludgeStr->Temp = DigestTemp;

// ESTIMATE COST DATA
Cost_AnaerobicDigestion( unitPtr );

// ESTIMATE ENVIRONMENTAL PROPERTIES OF EXITING STREAMS
Estimate_EnviroStreamProperties(ProdStr);
Estimate_EnviroStreamProperties(AirOutStr);
Estimate_EnviroStreamProperties(SludgeStr);
}

/***************************************************************************/
*   SOLVE BELT FILTER PRESS                                              *
*                                                                       *
/***************************************************************************/
Solve_Belt_Filter(LINK unitPtr)
{
  Belt_FilterPtr bfpPtr;
  St_CompPtr SludgeCompPtr, WashoutCompPtr, WashCompPtr, FeedCompPtr;
  St_LINK FeedStr, WashStr;
  St_LINK SludgeStr, WashoutStr, shead;
  St_LINK assStream[2];
  short i, StrNum;
  short StreamPoints;

double Tot_Sol_Cake; // Total solids in cake, weight percent
double Sludge_Rate; // Sludge-loading rate, kg/m-h
double Belt_Width; // Belt width, m
double WashFlow; // Washwater Flowrate, m3/m-h
double Sol_Recov; // Solids recovery, percent
double RemovFrac[compMax]; // Fraction transferred to sludge stream
double Sflow; // Entering stream mass flow rate, kg/h
double Wflow; // Entering Wash water flow rate, kg/h
double Fflow; // Filtrate stream mass flow rate, kg/h
double Cflow; // sludge Cake stream flow rate, kg/h
double Ssolid; // Feed stream solid mass flow rate, kg/h
double Fsolid; // Filtrate stream solid mass flow, kg/h
double Liq_In_Cake_Sludge; // Amount of liquid in the sludge cake
double LiqFrac; // Fraction of liquid in the feed stream
// that goes into sludge

double Liq_In_Feed; // kg/h
double RequiredWashFlow; // Washwater flowrate calculated by model, kg/h
short biomassIndex, waterIndex;
double intraInSludge, extraInSludge, totalInSludge;
double intraRemovalFrac;

double intraInSludgeBefore, extraInSludgeBefore;
double intraInSludgeAfter, extraInSludgeAfter;
double liquidInSludge, totalInSludgeAfter;

bfpPtr = (Belt_FilterPtr) unitPtr->theModel;

// FIND FEED AND PRODUCT STREAMS
shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );
StreamPoints = assStream[0]->NumOfPoints;
if( EqualPt(assStream[0]->spoints[StreamPoints-1], unitPtr->hpoints[0]))
{
  WashStr = assStream[0];
  FeedStr = assStream[1];
}
else
{
  WashStr = assStream[1];
FeedStr = assStream[0];
}
shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );
if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] ) )
{
    WashoutStr = assStream[0];
    SludgeStr = assStream[1];
}
else
{
    WashoutStr = assStream[1];
    SludgeStr = assStream[0];
}

// GET CURRENT DATA STRUCTURE VALUES
Tot_Sol_Cake = bfpPtr->Tot_Sol_Cake;
Sludge_Rate = bfpPtr->Sludge_Rate;
Belt_Width = bfpPtr->Belt_Width;
WashFlow = bfpPtr->WashFlow;

// FIND MASS FLOW OF FEED SLUDGE AND WASH WATER STREAMS, kg/h
Sflow = FeedStr->MassFlow; // Utilization factor should be introduced here in the future
Wflow = WashStr->MassFlow;

// CHECK IF FEED SLUDGE FLOWRATE IS ZERO
if( fabs(Sflow) < 0.0000001 )
{
    BioAlert(BIO_ALRT_OK, "The mass flow rate entering the Belt Filter is zero!");
    return;
}

// FIND SOLID FLOW RATE OF INPUT SLUDGE STREAM, kg/h
Ssolid = Estim_SolidMassFlow( FeedStr );

// CHECK IF THE SOLIDS CONCENTRATION IN FEED STREAM IS GREATER THAN THE CONCENTRATION IN THE DEWATERED SLUDGE STREAM
if( ((Ssolid/Sflow)*100.0 - Tot_Sol_Cake) > 1.0e-8)
{
    BioAlert(BIO_ALRT_OK, "The concentration of solids in the feed stream is greater than the desired concentration of solids in the dewatered sludge stream. Please check your feed stream data or increase the solids concentration in sludge!");
    return;
}

// CHECK IF BELT WIDTH IS SPECIFIED OR NOT
if(!bfpPtr->WidthSpec) // Estimate belt width
{
    Belt_Width = Ssolid / Sludge_Rate;
    bfpPtr->Belt_Width = Belt_Width;
}
else // Estimate solids-loading rate
{}
Sludge_Rate = Ssolid / Belt_Width;
  bfpPtr->Sludge_Rate = Sludge_Rate;
}

// ADJUST WASHWATER FLOWRATE, kg/h
waterIndex = GetComponentIndex( waterComp );
if( waterIndex != -1 )
  RequiredWashFlow = WashFlow*Belt_Width*theComps[waterIndex]->Density[0];
if( Wflow > 0.0000000001 )
  Multiply_Stream(WashStr, (RequiredWashFlow/Wflow));

// MASS BALANCES FOR THE SOLID COMPONENTS IN SLUDGE CAKE STREAM

// GET INDEX OF BIOMASS COMP AND REMOVAL FRAC OF INTRACELLULAR COMPS
Multiply_Stream(SludgeStr, 0.0);

intraRemovalFrac = 0.0;
biomassIndex = GetComponentIndex( biomassComp );
if( biomassIndex != -1 )
  intraRemovalFrac = bfpPtr->RemovFrac[biomassIndex];

FeedCompPtr = FeedStr->StCompList;
SludgeCompPtr = SludgeStr->StCompList;
for( i=0; i<NumOfComps; i++)
{
  intraRemovalFrac = MAX( intraRemovalFrac, bfpPtr->RemovFrac[i] );
  intraInSludge = FeedCompPtr->CompMassFlow * FeedCompPtr->CompLocation[0]* intraRemovalFrac;
  extraInSludge = FeedCompPtr->CompMassFlow * FeedCompPtr->CompLocation[1]* bfpPtr->RemovFrac[i];
  totalInSludge = intraInSludge + extraInSludge;
  SludgeCompPtr->CompMassFlow = totalInSludge;
  if( totalInSludge > 0.000001 )
  {
    SludgeCompPtr->CompLocation[0] = intraInSludge/totalInSludge;
    SludgeCompPtr->CompLocation[1] = extraInSludge/totalInSludge;
  }
  else
  {
    SludgeCompPtr->CompLocation[0] = 0.0;
    SludgeCompPtr->CompLocation[1] = 1.0;
  }
  FeedCompPtr = FeedCompPtr->next;
  SludgeCompPtr = SludgeCompPtr->next;
}
SludgeStr->MassFlow = Estim_MassFlow( SludgeStr );

// ESTIMATE THE AMOUNT OF LIQUID IN THE SLUDGE CAKE STREAM
Liq_In_Cake_Sludge = ((100.0 - Tot_Sol_Cake) / Tot_Sol_Cake) * Estim_SolidMassFlow( SludgeStr );
// ESTIMATE LIQUID IN THE FEED STREAM
Liq_In_Feed = FeedStr->MassFlow - Estim_SolidMassFlow( FeedStr );

// ESTIMATE FRACTION OF FEED LIQUID THAT GOES INTO THE SLUDGE STREAM
LiqFrac = Liq_In_Cake_Sludge / Liq_In_Feed;

if( LiqFrac > 1.0 )
    LiqFrac = 1.0;

if( LiqFrac < 0.0 )
    LiqFrac = 0.0;

// MASS BALANCES FOR LIQUID COMPONENTS AND EXTRACELLULAR FRACTION
FeedCompPtr = FeedStr->StCompList;
SludgeCompPtr = SludgeStr->StCompList;

for( i=0; i<NumOfComps; i++)
{
    if( bfpPtr->RemovFrac[i] < 0.0000001 )
    {
        intraInSludgeBefore = SludgeCompPtr->CompMassFlow * 
                               SludgeCompPtr->CompLocation[0];
        extraInSludgeBefore = SludgeCompPtr->CompMassFlow * 
                               SludgeCompPtr->CompLocation[1];

        liquidInSludge = FeedCompPtr->CompMassFlow * 
                         FeedCompPtr->CompLocation[1] * LiqFrac;

        intraInSludgeAfter = intraInSludgeBefore;
        extraInSludgeAfter = extraInSludgeBefore + liquidInSludge;
        totalInSludgeAfter = intraInSludgeAfter + extraInSludgeAfter;

        SludgeCompPtr->CompMassFlow = totalInSludgeAfter;
        if( totalInSludgeAfter > 0.000001 )
        {
            SludgeCompPtr->CompLocation[0] = 
                intraInSludgeAfter/totalInSludgeAfter;
            SludgeCompPtr->CompLocation[1] = 
                extraInSludgeAfter/totalInSludgeAfter;
        }
    }

    else
    {
        SludgeCompPtr->CompLocation[0] = 0.0;
        SludgeCompPtr->CompLocation[1] = 1.0;
    }
}

FeedCompPtr = FeedCompPtr->next;
SludgeCompPtr = SludgeCompPtr->next;

SludgeStr->MassFlow = Estim_MassFlow( SludgeStr );

// CALCULATE FLOWRATE AND COMPOSITION OF ***** STREAM
Subtract_Stream( FeedStr, SludgeStr, WashoutStr );
/** Solve_Spray_Dryer(): estimates the size and purchase cost of a Spray Dryer **/

Solve_Spray_Dryer(LINK unitPtr)
{
    Spray_DryerPtr spdPtr;
    St_CompPtr ProductCompPtr,AirinCompPtr, FeedCompPtr,AiroutCompPtr
    St_LINK FeedStr, AirinStr;
    St_LINK ProdStr,AiroutStr,shead;
    St_LINK assStream[2];
    short i, StrNum;
    short StreamPoints;
    short Number;       // Number of units
    double DryerDiameter; // Dryer Diameter, m
    double DryerVolume;  // Dryer Volume, m3
    double MaxDryerVolume; // Max. Dryer Volume, m3
    double MaxDryerDiameter; // Max. Dryer Diameter, m
    double EvaporationRate; // Overall Evaporation Rate, kg solvent/m3-h
    double DryingTemp;     // Drying temperature, °C
    double H_D_Ratio;      // Height/Diameter Ratio
    double DryerHeight;    // Dryer Height, m
    double SpecHeatRequirement; // Solvent Vaporization Heat, kcal/kg solvent
    double SpecPowerRequirement; // Power requirement per unit feed volume, kW/m3
    double NetDryingGasRequirement; // Drying Gas Requirement, kg gas/kg Solvent
    double RequiredDryingGas; // Total Drying Gas required, kg/h
    double factor; // Adjustment factor
    double solventRemoved, TotalSolventRemoved; // kg/h
    double intraSolventBefore, extraSolventBefore;
    double intraSolventAfter, extraSolventAfter;
    double totalSolventAfter;
spdPtr = (Spray_DryerPtr) unitPtr->theModel;

// FIND FEED AND PRODUCT STREAMS
shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );
StreamPoints = assStream[0]->NumOfPoints;
if( EqualPt(assStream[0]->spoints[StreamPoints-1], unitPtr->hpoints[0]))
{
    AirinStr = assStream[1];
    FeedStr  = assStream[0];
}
else
{
    AirinStr = assStream[0];
    FeedStr  = assStream[1];
}

shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );
if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] ) )
{
    ProdStr = assStream[0];
    AiroutStr = assStream[1];
}
else
{
    ProdStr = assStream[1];
    AiroutStr = assStream[0];
}

// CHECK IF FEED FLOWRATE IS ZERO
if( fabs(FeedStr->MassFlow) < 0.0000001 )
    return;

// GET VALUES FROM DATA STRUCTURE
DryerDiameter = spdPtr->DryerDiameter;
MaxDryerDiameter = spdPtr->MaxDryerDiameter;
EvaporationRate = spdPtr->EvaporationRate;
DryingTemp = spdPtr->DryingTemp;
H_D_Ratio = spdPtr->H_D_Ratio;
SpecHeatRequirement = spdPtr->SpecHeatRequirement;
SpecPowerRequirement = spdPtr->SpecPowerRequirement;
NetDryingGasRequirement = spdPtr->NetDryingGasRequirement;

// ESTIMATE TOTAL AMOUNT OF SOLVENT REMOVED
FeedCompPtr = FeedStr->StCompList;
TotalSolventRemoved = 0.0;
for(i=0; i<NumOfComps; i++)
{
    if( spdPtr->EvapFrac[i] > 0.000001 )
    {
        TotalSolventRemoved += FeedCompPtr->CompMassFlow* spdPtr->EvapFrac[i];
        FeedCompPtr = FeedCompPtr->next;
    }
}
// ASSIGN DRYER CAPACITY (kg/h) TO SIZE POINTER
unitPtr->Size = TotalSolventRemoved;

// CALCULATE REQUIRED DRYING MEDIUM, kg/h
RequiredDryingGas = NetDryingGasRequirement*TotalSolventRemoved;

// CALCULATE ADJUSTMENT FACTOR
factor = 1.0;
if( AirinStr->MassFlow > 1.0e-8 )
    factor = RequiredDryingGas/AirinStr->MassFlow;

// ESTIMATE ADJUSTED NEUTRALIZING STREAM
Multiply_Stream(AirinStr, factor);

// SIZING CALCULATIONS
if(unitPtr->m_isRatingMode) {
    DryerVolume = H_D_Ratio*(PI/4.0)*pow(DryerDiameter,3.0);
    EvaporationRate = TotalSolventRemoved/(DryerVolume * unitPtr->Number);
} else {
    MaxDryerVolume = H_D_Ratio*(PI/4.0)*pow(MaxDryerDiameter,3.0);
    DryerVolume = TotalSolventRemoved/EvaporationRate;
    Number = 1;
    if(DryerVolume > MaxDryerVolume)
    {
        Number = (short) ceil( DryerVolume/MaxDryerVolume );
        DryerVolume /= Number;
    }

    DryerDiameter = pow((4.0*DryerVolume/PI/H_D_Ratio),(1.0/3.0));
    unitPtr->Number = Number;
}

// CALCULATE DRUM LENGTH, m
DryerHeight = DryerDiameter*H_D_Ratio;

// ASSIGN VALUES TO DATA STRUCTURE
spdPtr->DryerDiameter = DryerDiameter;
spdPtr->DryerHeight = DryerHeight;
spdPtr->EvaporationRate = EvaporationRate;

// COPY FEED STREAM INTO OUTPUT STREAMS
Copy_Stream( AirinStr, AiroutStr );
Copy_Stream( FeedStr, ProdStr );

// CONSIDER SOLVENT REMOVAL BY VAPORIZATION
ProductCompPtr = ProdStr->StCompList;
AiroutCompPtr = AiroutStr->StCompList;
for(i=0; i<NumOfComps; i++)
{   if( spdPtr->EvapFrac[i] > 0.000001 )
    {   

solventRemoved = ProductCompPtr->CompMassFlow * spdPtr->EvapFrac[i];

intraSolventBefore = ProductCompPtr->CompMassFlow *
ProductCompPtr->CompLocation[0];
extraSolventBefore = ProductCompPtr->CompMassFlow *
ProductCompPtr->CompLocation[1];

if( solventRemoved < extraSolventBefore )
{
    intraSolventAfter = intraSolventBefore;
    extraSolventAfter -= solventRemoved;
}
else
{
    extraSolventAfter = 0.0;
    intraSolventAfter -= (solventRemoved - extraSolventBefore);
}

ProductCompPtr->CompMassFlow -= solventRemoved;
AiroutCompPtr->CompMassFlow += solventRemoved;
totalSolventAfter = intraSolventAfter + extraSolventAfter;

if( fabs(totalSolventAfter) > 0.000001 )
{
    ProductCompPtr->CompLocation[0] =
intraSolventAfter / totalSolventAfter;
    ProductCompPtr->CompLocation[1] =
extraSolventAfter / totalSolventAfter;
}
else
{
    ProductCompPtr->CompLocation[0] = 0.0;
    ProductCompPtr->CompLocation[1] = 1.0;
}

AiroutCompPtr = AiroutCompPtr->next;
ProductCompPtr = ProductCompPtr->next;

// CALCULATE HEATING DUTY, kcal/h
if( !unitPtr->m_isHeatingDutySet )
    unitPtr->Duty[0] = TotalSolventRemoved*SpecHeatRequirement;

// CALCULATE POWER REQUIREMENT, kW
if( !unitPtr->m_isPowerSet )
    unitPtr->Power = Estim_VolFlow(FeedStr)*SpecPowerRequirement;

// ASSIGN DRYING TEMPERATURE TO EXIT STREAM TEMP
ProdStr->Temp = DryingTemp;
AiroutStr->Temp = DryingTemp;

// ESTIMATE PURCHASE COST OF TRAY DRYER
Cost_Spray_Dryer( unitPtr );
// ESTIMATE ENVIRONMENTAL PROPERTIES OF THE EXIT STREAMS
Estimate_EnviroStreamProperties(ProdStr);
Estimate_EnviroStreamProperties(AiroutStr);
}

/********************************************
* Solve_Rotary_Dryer(): estimates the size and purchase *
* cost of a Rotary Dryer *
********************************************/
Solve_Rotary_Dryer(LINK unitPtr)
{
    Rotary_DryerPtr rodPtr;
    St_CompPtr ProductCompPtr,AirinCompPtr, FeedCompPtr,AiroutCompPtr
    St_LINK FeedStr, AirinStr;
    St_LINK ProdStr,AiroutStr,shead;
    St_LINK assStream[2];
    short i, StrNum;
    short StreamPoints;
    short Number; // Number of units
    double DrumDiameter; // Drum Diameter, m
    double DrumVolume; // Drum Volume, m³
    double MaxDrumVolume; // Max. Drum Volume, m³
    double MaxDrumDiameter; // Max. Drum Diameter, m
    double EvaporationRate; // Overall Evaporation Rate, kg solvent/m³-h
    double DryingTemp; // Drying temperature, °C
    double L_D_Ratio; // Length/Diameter Ratio
    double DrumLength; // Drum Length, m
    double SpecHeatRequirement; // Solvent Vaporization Heat, kcal/kg solvent
    double SpecPowerRequirement; // Power requirement per unit tray area, kW/m²
    double NetDryingGasRequirement; // Net Drying Gas Requirement, kg gas/m²-s
    double RequiredDryingGas; // Total Drying Gas required, kg/h
    double factor; // Adjustment factor
    double solventRemoved, TotalSolventRemoved; // kg/h
    double intraSolventBefore, extraSolventBefore;
    double intraSolventAfter, extraSolventAfter;
    double totalSolventAfter;

    rodPtr = (Rotary_DryerPtr) unitPtr->theModel;

    // FIND FEED AND PRODUCT STREAMS
    shead = stream_head;
    Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );
    StreamPoints = assStream[0]->NumOfPoints;
    if( EqualPt(assStream[0]->spoints[StreamPoints-1], unitPtr->hpoints[0])} { AirinStr = assStream[0]; FeedStr = assStream[1]; }
else { AirinStr = assStream[1]; FeedStr = assStream[0]; }
shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );
if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] ) )
{
    ProdStr = assStream[0];
    AiroutStr = assStream[1];
}
else
{
    ProdStr = assStream[1];
    AiroutStr = assStream[0];
}

// CHECK IF FEED FLOWRATE IS ZERO
if( fabs(FeedStr->MassFlow) < 0.0000001 )
    return;

// GET VALUES FROM DATA STRUCTURE
DrumDiameter = rodPtr->DrumDiameter;
MaxDrumDiameter = rodPtr->MaxDrumDiameter;
EvaporationRate = rodPtr->EvaporationRate;
DryingTemp = rodPtr->DryingTemp;
L_D_Ratio = rodPtr->L_D_Ratio;
SpecHeatRequirement = rodPtr->SpecHeatRequirement;
SpecPowerRequirement = rodPtr->SpecPowerRequirement;
NetDryingGasRequirement = rodPtr->NetDryingGasRequirement;

// ESTIMATE TOTAL AMOUNT OF SOLVENT REMOVED
FeedCompPtr = FeedStr->StCompList;
TotalSolventRemoved = 0.0;
for(i=0; i<NumOfComps; i++)
{
    if( rodPtr->EvapFrac[i] > 0.0000001 )
        TotalSolventRemoved += FeedCompPtr->CompMassFlow*
                                rodPtr->EvapFrac[i];
    FeedCompPtr = FeedCompPtr->next;
}

// SIZING CALCULATIONS
if(unitPtr->m_isRatingMode)
{
    DrumVolume = L_D_Ratio*(PI/4.0)*pow(DrumDiameter,3.0);
    EvaporationRate = TotalSolventRemoved/(DrumVolume * unitPtr->Number);
}
else
{
    MaxDrumVolume = L_D_Ratio*(PI/4.0)*pow(MaxDrumDiameter,3.0);
    DrumVolume = TotalSolventRemoved/EvaporationRate;

    Number = 1;
    if(DrumVolume > MaxDrumVolume)
    {
        Number = (short) ceil( DrumVolume/MaxDrumVolume );
    }
DrumVolume /= Number;
}

DrumDiameter = pow((4.0*DrumVolume/PI/L_D_Ratio),(1.0/3.0));
unitPtr->Number = Number;

// CALCULATE DRUM LENGTH, m
DrumLength = DrumDiameter*L_D_Ratio;

// ASSIGN VALUES TO DATA STRUCTURE
rodPtr->DrumDiameter = DrumDiameter;
rodPtr->DrumLength = DrumLength;
rodPtr->EvaporationRate = EvaporationRate;

// CALCULATE REQUIRED DRYING MEDIUM, kg/h
RequiredDryingGas = (PI*pow(DrumDiameter,2.0)/4.0)*
                     NetDryingGasRequirement*3600.0;

// CALCULATE ADJUSTMENT FACTOR
factor = 1.0;
if( AirinStr->MassFlow > 1.0e-8 )
    factor = RequiredDryingGas/AirinStr->MassFlow;

// INDIRECT HEATING - NO AIR INLET
if (!rodPtr->Direct)
    factor = 0.0;

// ESTIMATE ADJUSTED NEUTRALIZING STREAM
Multiply_Stream(AirinStr, factor);

// COPY FEED STREAM INTO OUTPUT STREAMS
Copy_Stream( AirinStr, AiroutStr );
Copy_Stream( FeedStr, ProdStr );

// CONSIDER SOLVENT REMOVAL BY VAPORIZATION
ProductCompPtr = ProdStr->StCorrpList;
AiroutCompPtr = AiroutStr->StCompList;
for(i=0; i<NumOfComps; i++)
{
    if( rodPtr->EvapFrac[i] > 0.000001 )
    {
        solventRemoved = ProductCompPtr->CompMassFlow *
                         rodPtr->EvapFrac[i];

        intraSolventBefore = ProductCompPtr->CompMassFlow *
                             ProductCompPtr->CompLocation[0];
        extraSolventBefore = ProductCompPtr->CompMassFlow *
                             ProductCompPtr->CompLocation[1];

        if( solventRemoved < extraSolventBefore )
        {
            intraSolventAfter = intraSolventBefore;
            extraSolventAfter -= solventRemoved;
        }
    }
    else

{  
    extraSolventAfter = 0.0;  
    intraSolventAfter -= (solventRemoved - extraSolventBefore);  
}

ProductCompPtr->CompMassFlow -= solventRemoved;  
AiroutCompPtr->CompMassFlow += solventRemoved;  

totalSolventAfter = intraSolventAfter + extraSolventAfter;

if( fabs(totalSolventAfter) > 0.000001 )
{
    ProductCompPtr->CompLocation[0] = intraSolventAfter / totalSolventAfter;  
    ProductCompPtr->CompLocation[1] = extraSolventAfter / totalSolventAfter;
}
else
{
    ProductCompPtr->CompLocation[0] = 0.0;  
    ProductCompPtr->CompLocation[1] = 1.0;
}

AiroutCompPtr = AiroutCompPtr->next;  
ProductCompPtr = ProductCompPtr->next;

// CALCULATE HEATING DUTY, kcal/h  
if( !unitPtr->m_isHeatingDutySet )
    unitPtr->Duty[0] = TotalSolventRemoved*SpecHeatRequirement;

// CALCULATE POWER REQUIREMENT, kW  
if( !unitPtr->m_isPowerSet )
    unitPtr->Power = (PI*DrumDiameter*DrumLength)*SpecPowerRequirement;

// ASSIGN DRYING TEMPERATURE TO EXIT STREAM TEMP  
ProdStr->Temp = DryingTemp;  
AiroutStr->Temp = DryingTemp;

// ESTIMATE PURCHASE COST OF TRAY DRYER  
Cost_Rotary_Dryer( unitPtr );

// ESTIMATE ENVIRONMENTAL PROPERTIES OF THE EXIT STREAMS  
Estimate_EnviroStreamProperties(ProdStr);  
Estimate_EnviroStreamProperties(AiroutStr);

 /******************************************************************  
• SOLVE ABSORPTION COLUMN  
*  
* Programmed by: Costas Abeliotis & DP  
* Date: July 27, 1992  
*/
Solve_Absorption(LINK unitPtr)
{
    AbsorptionPtr absPtr;
    St_CompPtr GCompPtr, LCompPtr;
    St_LINK Gas_In_Str, Gas_Out_Str;
    St_Link Liq_In_Str, Liq_Out_Str;
    St_Link shead, assStream[2];
    short i, StrNum, StreamPoints;
    double Cf, LiqVisc, GasVisc;
    double Liq_Flow, Gas_Flow;
    double Liq_Dens, Gas_Dens;
    double m_xAxisValue, m_yAxisValue;
    double G_Prime, L_Prime;
    double S_Area;
    double AmountRemoved;
    short designCompIndex;
    short m_operationMode;
    double m_unitHeightDeltaP;
    double m_columnHeight;
    double m_deltaP;
    double m_diameter;
    double m_diamLeft, m_diamRight, m_diamMiddle;
    double fLeft, fRight, fMiddle;
    short counter;

    absPtr = (AbsorptionPtr) unitPtr->theModel;

    // FIND FEED AND PRODUCT STREAMS
    shead = stream_head;
    Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );

    StreamPoints = assStream[0]->NumOfPoints;
    if( EqualPt( assStream[0]->spoints[StreamPoints-1], unitPtr->hpoints[0] ) )
    {
        Liq_In_Str = assStream[0];
        Gas_In_Str = assStream[1];
    }
    else
    {
        Liq_In_Str = assStream[1];
        Gas_In_Str = assStream[0];
    }

    shead = stream_head;
    Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );

    if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] ) )

{ 
    Liq_Out_Str = assStream[0];
    Gas_Out_Str = assStream[1];
}
else
{
    Liq_Out_Str = assStream[1];
    Gas_Out_Str = assStream[0];
}

// CHECK IF FEED FLOWRATE IS ZERO
if( Gas_In_Str->MassFlow < 0.0000000001 )
    return;

// GET DATA_STRUCTURE VALUES
Cf = absPtr->Cf;
LiqVisc = absPtr->LiqVisc / 1000.0; // from cp to kg/m-s
GasVisc = absPtr->GasVisc / 1000.0;
m_operationMode = absPtr->m_operationMode;

// CALCULATE MATERIAL BALANCES
Copy_Stream( Gas_In_Str, Gas_Out_Str );
Copy_Stream( Liq_In_Str, Liq_Out_Str );

GCompPtr = Gas_Out_Str->StCompList;
LCompPtr = Liq_Out_Str->StCompList;
for(i=0; i<NumOfComps; i++)
{
    AmountRemoved = GCompPtr->CompMassFlow * absPtr->RemovFrac[i];
    GCompPtr->CompMassFlow *= 1 - absPtr->RemovFrac[i];
    LCompPtr->CompMassFlow += AmountRemoved;
    GCompPtr = GCompPtr->next;
    LCompPtr = LCompPtr->next;
}

Gas_Out_Str->MassFlow = Estim_MassFlow( Gas_Out_Str );
Liq_Out_Str->MassFlow = Estim_MassFlow( Liq_Out_Str );

// COLUMN SIZING
Liq_Flow = Liq_In_Str->MassFlow / 3600; // from kg/h to kg/sec
Gas_Flow = Gas_In_Str->MassFlow / 3600;

Liq_Dens = Stream_Density( Liq_In_Str);
Gas_Dens = Stream_Density( Gas_In_Str );

// FIND THE VALUE OF THE X-AXIS IN FIGURE 6.34 TREYBAL
m_xAxisValue = Liq_Flow/Gas_Flow * pow( Gas_Dens/(Liq_Dens - Gas_Dens), 0.5 );

if(m_operationMode == 0) // Column Diameter specified
{
    // GET DIAMETER FROM DATA STRUCTURE
    m_diameter = absPtr->Diam;
    S_Area = PI*pow(m_diameter, 2.0)/4.0;
}
G_Prime = Gas_Flow / S_Area;

// FIND THE VALUE OF THE Y-AXIS IN FIGURE 6.34 TREYBAL
m_yAxisValue = pow(G_Prime, 2.0)*Cf*pow(LiqVisc, 0.1)/(Gas_Dens*(Liq_Dens - Gas_Dens));

// CALCULATE UNIT HEIGHT PRESSURE DROP
absPtr->m_unitHeightDeltaP = GetUnitHeightDeltaP(m_xAxisValue, m_yAxisValue);

// CALCULATE COLUMN HEIGHT
absPtr->Height = CalculateAbsorberHeight(unitPtr, m_diameter);

// CALCULATE TOTAL PRESSURE DROP
absPtr->DeltaPres = CalculateAbsorberPressureDrop(unitPtr, m_diameter);

} else if(m_operationMode == 2) // DP/Z specified {
  // GET DIAMETER FROM DATA STRUCTURE
  m_unitHeightDeltaP = absPtr->m_unitHeightDeltaP;

  // CALCULATE Y-AXIS VALUE
  m_yAxisValue = GetYAxisValue(m_xAxisValue, m_unitHeightDeltaP);

  // CALCULATE COLUMN DIAMETER
  m_diameter = sqrt(4.0*Gas_Flow / (PI*G_Prime));
  absPtr->Diam = m_diameter;

  // CALCULATE COLUMN HEIGHT
  absPtr->Height = CalculateAbsorberHeight(unitPtr, m_diameter);

  // CALCULATE TOTAL PRESSURE DROP
  absPtr->DeltaPres = CalculateAbsorberPressureDrop(unitPtr, m_diameter);

} else // Total DP specified {
  // GET THE GIVEN TOTAL PRESSURE DROP
  m_deltaP = absPtr->DeltaPres;

  // USE BISECTION METHOD TO FIND THE DIAMETER THAT MATCHES THE GIVEN DP
  m_diamLeft = 0.001; m_diamRight = 50.0;
  fLeft = CalculateAbsorberPressureDrop(unitPtr, m_diamLeft) - m_deltaP;
  fRight = CalculateAbsorberPressureDrop(unitPtr, m_diamRight) - m_deltaP;

  counter = 0;
  if( fLeft * fRight < 0 ) {
    while (counter<300 && fabs(m_diamLeft - m_diamRight) > 0.001) {
      m_diamMiddle = (m_diamLeft + m_diamRight)/2.0;
fMiddle = CalculateAbsorberPressureDrop(unitPtr, m_diamMiddle) - m_deltaP;

if (fMiddle * fLeft < 0)
    m_diamRight = m_diamMiddle;
else
    m_diamLeft = m_diamMiddle;

fLeft = fMiddle;

counter++;

else
{
    BioAlert(BIO_ALRT_OK, "No column diameter value could be found in the range [0,50] m ");
    return;
}

// WHEN SOLUTION REACHED
m_diameter = m_diamMiddle;

S_Area = PI*pow(m_diameter, 2.0)/4.0;
G_Prime = Gas_Flov / S_Area;

// FIND THE VALUE OF THE Y-AXIS IN FIGURE 6.34 TREYBAL
m_yAxisValue = pow(G_Prime, 2.0)*Cf*pow(LiqVisc, 0.1)/
   (Gas_Dens*(Liq_Dens - Gas_Dens));

// CALCULATE UNIT HEIGHT PRESSURE DROP
absPtr->m_unitHeightDeltaP = GetUnitHeightDeltaP(m_xAxisValue, m_yAxisValue);

// CALCULATE COLUMN HEIGHT
absPtr->Height = CalculateAbsorberHeight(unitPtr, m_diameter);

// ESTIMATE PURCHASE COST OF ABSORPTION COLUMN
Cost_Absorption( unitPtr );

// ESTIMATE ENVIRONMENTAL PROPERTIES OF EXITING STREAMS
Estimate_EnviroStreamProperties(Liq_Out_Str);
Estimate_EnviroStreamProperties(Gas_Out_Str);

double CalculateAbsorberPressureDrop(LINK unitPtr, double m_diameter)
{
AbsorptionPtr absPtr;
St_CompPtr CompPtr GComoPtr, LCompPtr;
St_LINK Gas_In_Str, Gas_Out_Str;
St_LINK Liq_In_Str, Liq_Out_Str;
St_LINK shead, assStream[2];
short i, StrNum, StreamPoints;
double LiqVisc, Cf;
double Liq_Flow, Gas_Flow;
double Liq_Dens, Gas_Dens;
double m_xAxisValue, m_yAxisValue;
double G_Prime;
double S_Area;
double m_columnHeight;
double m_unitHeightDeltaP;

absPtr = (AbsorptionPtr) unitPtr->theModel;

// FIND FEED AND PRODUCT STREAMS
shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );
StreamPoints = assStream[0]->NumOfPoints;
if( EqualPt( assStream[0]->spoints[StreamPoints-1], unitPtr->hpoints[0] ) )
{
    Liq_In_Str = assStream[0];
    Gas_In_Str = assStream[1];
}
else
{
    Liq_In_Str = assStream[1];
    Gas_In_Str = assStream[0];
}

shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );
if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] ) )
{
    Liq_Out_Str = assStream[0];
    Gas_Out_Str = assStream[1];
}
else
{
    Liq_Out_Str = assStream[1];
    Gas_Out_Str = assStream[0];
}

// GET DATA_STRUCTURE VALUES
LiqVisc = absPtr->LiqVisc / 1000.0; // from cp to kg/m-s
Cf = absPtr->Cf;

// GET STREAM DENSITIES
Liq_Dens = Stream_Density( Liq_In_Str);
Gas_Dens = Stream_Density( Gas_In_Str );

// CALCULATE G_Prime
Liq_Flow = Liq_In_Str->MassFlow / 3600;
Gas_Flow = Gas_In_Str->MassFlow / 3600;
S_Area = PI*pow(m_diameter, 2.0)/4.0;
G_Prime = Gas_Flow / S_Area;

// FIND THE VALUE OF THE X-AXIS IN FIGURE 6.34 TREYBAL
m_xAxisValue = Liq_Flow/Gas_Flow * pow( Gas_Dens/(Liq_Dens - Gas_Dens), 0.5 );
// FIND THE VALUE OF THE Y-AXIS IN FIGURE 6.34 TEREYBAL
m_yAxisValue = pow(G_Prime, 2.0)*Cf*pow(LiqVisc, 0.1)/
   (Gas_Dens*(Liq_Dens - Gas_Dens));

// CALCULATE UNIT HEIGHT PRESSURE DROP
m_unitHeightDeltaP = GetUnitHeightDeltaP(m_xAxisValue, m_yAxisValue);

// CALCULATE COLUMN HEIGHT
m_columnHeight = CalculateAbsorberHeight(unitPtr, m_diameter);
return (m_unitHeightDeltaP*m_columnHeight);
}

double CalculateAbsorberHeight(LINK unitPtr, double m_diameter)
{
    AbsorptionPtr absPtr;
    St_CompPtr GCompPtr, LCompPtr;
    St_LINK Gas_In_Str, Gas_Out_Str;
    St_LINK Liq_In_Str, Liq_Out_Str;
    St_LINK shead, assStream[2];
    short i, StrNum, StreamPoints;
    double var1;
    double LiqVisc, GasVisc;
    double dp,atot,awet,kL,kG,KGa;
    double Re,Fr,We;
    double sigmaLiq,sigmaCrit;
    double Diff_L, Diff_G, m;
    double Liq_Flow, Gas_Flow;
    double Liq_Dens, Gas_Dens;
    double m_xAxisValue, m_yAxisValue;
    double G_Prime, L_Prime;
    double S_Area;
    double HTU, NTU;
    double A_Factor;
    double y1, y2, x2;
    double R = 82.06e-5; // Universal gas constant, m3-atm/mole-K
    short designCompIndex;
    double m_unitHeightDeltaP;

    absPtr = (AbsorptionPtr) unitPtr->theModel;

    // FIND FEED AND PRODUCT STREAMS
    shead = stream_head;
    Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );

    StreamPoints = assStream[0]->NumOfPoints;
    if( EqualPt( assStream[0]->spoints[StreamPoints-1], unitPtr->hpoints[0] ) )
    {
        Liq_In_Str = assStream[0];
        Gas_In_Str = assStream[1];
    }
    else
    {
        Liq_In_Str = assStream[1];
        Gas_In_Str = assStream[0];
    }
shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );

if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2]) )
{
    Liq_Out_Str = assStream[0];
    Gas_Out_Str = assStream[1];
}
else
{
    Liq_Out_Str = assStream[1];
    Gas_Out_Str = assStream[0];
}

// GET DATA_STRUCTURE VALUES
LiqVisc = absPtr->LiqVisc / 1000.0; // from cp to kg/m-s
GasVisc = absPtr->GasVisc / 1000.0;
atot = absPtr->atot;
sigmaCrit = absPtr->sigmaCrit/1000.0; // from dyn/cm to N/m
sigmaLiq = absPtr->sigmaLiq/1000.0; // from dyn/cm to N/m
dp = absPtr->dp;
Diff_L = absPtr->Diff_L / 10000.0; // from cm2/sec to m2/sec
Diff_G = absPtr->Diff_G / 10000.0;

// GET HENRY'S LAW CONSTANT FOR DESIGN COMPONENT
designCompIndex = GetComponentIndex(absPtr->DesignComp);
m = theComps[designCompIndex]->Henry;
m  /= R*298.15; // Convert Henry's constant from atm-m3/mole to dimensionless

// CALCULATE G_Prime, L_Prime
Liq_Flow = Liq_In_Str->MassFlow / 3600; // from kg/h to kg/sec
Gas_Flow = Gas_In_Str->MassFlow / 3600;

Liq_Dens = Stream_Density( Liq_In_Str);
Gas_Dens = Stream_Density( Gas_In_Str );

S_Area = PI*pow(m_diameter, 2.0)/4.0;
L_Prime = Liq_Flow / S_Area;
G_Prime = Gas_Flow / S_Area;

// ESTIMATE COLUMN HEIGHT
Re = L_Prime/atot/LiqVisc;
Fr = (pow(L_Prime,2)*atot)/(pow(Liq_Dens,2)*9.81);
We = pow(L_Prime,2)/((Liq_Dens*sigmaLiq*atot);

// ONDA CORRELATIONS
awet = atot*(1-exp(-1.45*pow((sigmaLiq/sigmaCrit),0.75)*pow(Re,0.1)*
pow(Fr,-0.05)*pow(We,0.2)));

kG = atot*Diff_G*5.23*pow((G_Prime/atot/GasVisc),0.7)*
    pow((GasVisc/Gas_Dens/Diff_G),0.33)*pow((atot*dp),-2);

kL = (0.0051*pow((L_Prime/awet/LiqVisc),0.66)*pow((LiqVisc/Liq_Dens/Diff_L),
    -0.5)*pow((atot*dp),-0.4))/pow((Liq_Dens/LiqVisc/9.81),0.33);
\[ KGa = \frac{1.0}{\left( \frac{1}{(kG*awet)} + \frac{m}{(kL*awet)} \right)}; \]

\[ HTU = \frac{G_{\text{Prime}}/\text{Gas\_Dens}}{KGa}; \quad \text{// Height of an absorbing transfer unit} \]

\[ \text{// ESTIMATE NUMBER OF TRANSFER UNITS} \]
\[ \text{Liq\_Flow} = \text{Estim\_MoleFlow}(\text{Liq\_In\_Str}); \]
\[ \text{Gas\_Flow} = \text{Estim\_MoleFlow}(\text{Gas\_In\_Str}); \]
\[ \text{A\_Factor} = \frac{\text{Liq\_Flow}}{m} / \frac{\text{Gas\_Flow}}{}; \]

\[ \text{if( A\_Factor < 1.0 )} \]
\[ \{
\text{    BioAlert(BIO\_ALRT\_OK, "Please increase the liquid flowrate or reduce the gas flowrate of the absorber so that the unit can function properly!");}
\text{    A\_Factor = 1.00000001;}
\} \]
\[ \text{y1} = \text{Comp\_MoleFrac}(\text{Gas\_In\_Str}, \text{absPtr->DesignComp}); \]
\[ \text{y2} = \text{Comp\_MoleFrac}(\text{Gas\_Out\_Str}, \text{absPtr->DesignComp}); \]
\[ \text{x2} = \text{Comp\_MoleFrac}(\text{Liq\_In\_Str}, \text{absPtr->DesignComp}); \]

\[ \text{if((y2 - m*x2) < 0.0)} \]
\[ \text{NTU} = 0.0; \]
\[ \text{else} \]
\[ \{
\text{    var1 = \frac{(y1 - m*x2)/(y2 - m*x2) * (1 - 1/A\_Factor) + 1/A\_Factor}{(1 - 1/A\_Factor)};}
\text{    NTU = \frac{\log(var1)}{1 - 1/A\_Factor};}
\} \]

\[ \text{// ESTIMATE COLUMN HEIGHT} \]
\[ \text{return (NTU * HTU);} \]

/**************************************************************************
* SOLVE CYCLONE
* Programmed by: Kostas Abeliotis
* Date: 1/25/1993
* Reference 1: Introduction to Industrial Gas Cleaning,F.A Dulienn
* Paragraph 3.5 pp58 - 86
* Reference 2: The collection efficiency of cyclone type particle
* collectors - A new theoretical approach
* Leith & Licht, AIChE Symposium Series, Vol 68, No 126
* Air Pollution and Its Control, pp 196 - 206
**************************************************************************/
Solve_Cyclone(LINK unitPtr)
{
    CyclonePtr cycPtr;
    St\_CompPtr CleanCompPtr, DustCompPtr, FeedCompPtr;
    St\_LINK FeedStr, DustStr;
    St\_LINK CleanStr, shead;
    St\_LINK assStream[2];
    short i, StrNum;
    short StreamPoints;
double PartDens; /* Particle density, kg/m³ */
double vel; /* Inlet gas velocity, m/s */
double visc; /* Inlet gas viscosity, cp */
double dens; /* Inlet gas density, kg/m³ */
da; /* Inlet Height, m */
b; /* Inlet Width, m */
D; /* Cyclone body diameter, m */
De; /* Gas outlet diameter, m */
h; /* Height of upper cylindrical body, m */
S; /* Outlet Length, m */
H; /* Overall Cyclone Height, m */
B; /* Outlet duct diameter for dust, m */
l; /* Natural Length, m */
d; /* Diameter of cyclone at point where vortex turns, m */
Vnl; /* Volume of cyclone at natural length, m³ */
VH; /* Volume of cyclone below exit duct, m³ */
Vs; /* Annular shaped volume above exit duct, m³ */
Ka; 
Kb; 
Kc; /* Cyclone volume constant */
C; /* Cyclone Design Number */
gamma; /* Vortex parameter */
DP; /* Pressure drop, KPa */
DPmax; /* Maximum allowable Pressure drop, KPa */
mflow; /* Entering mass flow rate, kg/h */
Q; /* Feed volumetric flowrate, m³/h */
AreaIn; /* Inlet Cross sectional Area, m² */
psi; /* Cyclone inertial impaction parameter */
Eff; /* Collection Efficiency for each size range */
Efficiency; /* Overall Collection Efficiency */
AmountRemoved; /* Amount of particles removed, kg/h */
num,den; /* Dummy variables */
term1,term2,term3,term4,term5; /* Dummy variables */
RemovFrac[compMax]; /* Removal Fraction for each component */
MeanPartSize[compMax]; /* Particle diameter, μm */
Fraction_In_Range[compMax]; /* Fraction of particles in range */
cycPtr = (CyclonePtr) unitPtr->theModel;

/* FIND FEED AND PRODUCT STREAMS */
shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );
FeedStr = assStream[0];

shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );

if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] ) )
{
    CleanStr = assStream[0];
    DustStr = assStream[1];
}
else
{
    CleanStr = assStream[1];
    DustStr = assStream[0];
}
/* GET CURRENT DATA STRUCTURE VALUES */
for(i=0; i<10; i++)
{
    MeanPartSize[i] = (cycPtr->MeanPartSize[i])*1.0e-6;    // Convert µm to n
    Fraction_In_Range[i] = cycPtr->Fraction_In_Range[i];
    RemovFrac[i] = cycPtr->RemovFrac[i];
}
visc = (cycPtr->visc)*1.0e-3*3600.0;    /* Convert cp to (Kg/m-h) */
vel = (cycPtr->vel)*3600.0;    /* Convert m/s to m/h */
PartDens = cycPtr->PartDens;
dens = cycPtr->dens;
DPmax = (cycPtr->DPmax)*101.33;    /* Convert bar to kPa */

/* CHECK IF FEED MASS FLOWRATE IS ZERO */
mflow = FeedStr->MassFlow;
if( fabs(mflow) < 0.0000001 )
{
    BioAlert(BIO_ALRT_OK, "The mass flowrate of the feed stream to the
    'Cyclone' is zero. Please increase it so that the program can function
    properly !");
    return;
}

// INITIALIZE NUMBER OF UNITS
unitPtr->Number = 1;

do {
    /* ESTIMATE VOLUMETRIC FLOWRATE OF FEED STREAM, m³/h */
    Q = Estim_VolFlow(FeedStr)/unitPtr->Number/unitPtr->UtilFactor;

    /* CALCULATE CROSS SECTIONAL AREA OF INLET DUCT, m² */
    AreaIn = Q/vel;

    /* CALCULATE BODY DIAMETER, m */
    D = pow(8.0*AreaIn,0.5);
    cycPtr->D = D;

    /* CALCULATE INLET HEIGHT, m */
    a = D/2.0;
    cycPtr->a = a;

    /* CALCULATE INLET WIDTH, m */
    b = D/4.0;
    cycPtr->b = b;

    /* CALCULATE OUTLET GAS DIAMETER, m */
    De = D/2.0;
    cycPtr->De = De;

    /* CALCULATE HEIGHT OF THE UPPER CYLINDRICAL BODY, m */
    h = 2.0*D;
    cycPtr->h = h;
/* CALCULATE OUTLET DUCT LENGTH, m */
S = 5.0*D/8.0;
cycPtr->S = S;

/* CALCULATE OVERALL HEIGHT, m */
H = 4.0*D;
cycPtr->H = H;

/* CALCULATE OUTLET DUCT DIAMETER, m */
B = D/4.0;
cycPtr->B = B;

/* CALCULATE PRESSURE DROP, Pa */
DP = 16*(a*b)/pow(De,2.0)*pow(cycPtr->vel,2.0)*(dens/2.0);

if( (DP/101325.0) > DPmax )
    unitPtr->Number += 1;

cycPtr->DP = DP/(1.0E5); /* Convert Pa to bar */

while (cycPtr->DP > DPmax );

unitPtr->Size = Q;

/* ALGORITHM FOR CALCULATION OF THE OVERALL EFFICIENCY BY THE SYSTEM */
if(!cycPtr->UserSpecEff)
{
    /* CALCULATE THE NATURAL LENGTH, m */
l = 2.3 * De *pow((pow(D,2.0)/AreaIn),(1.0/3.0));

    /* CALCULATE Vs */
Vs = 3.14159*(S - (a/2.0))*(pow(D,2.0) - pow(De,2.0))/4.0;

    /* COMPARE THE NATURAL LENGTH WITH THE DIMENSION (H-S) */
if(l < (H-S))
{
    d = D - (D - B)*((S + l - h)/(H - h));
    term1 = (3.14159*pow(D,2.0)/4.0)*(h - S);
    term2 = ((3.14159*pow(D,2.0)*
        (1 + S - h))/12.0)*(1.0+(d/D)+pow((d/D),2.0));
    term3 = (3.14159*pow(De,2.0)/4.0)*1;
    Vnl = term1 + term2 - term3;
    Kc = (Vs + (Vnl/2.0))/pow(D,3.0);
}
else
{
    term1 = (3.14159*pow(D,2.0)/4.0)*(h - S);
    term4 = (3.14159*pow(D,2.0)/4.0)*
        ((H - h)/3.0)*(1.0+(d/D)+pow((d/D),2.0));
    term5 = (3.14159*pow(De,2.0)/4.0)*(H - S);

    Kc = Vs/(1.0+(d/D)+pow((d/D),2.0));
}
}
\[ VH = \text{term1} + \text{term4} - \text{term5}; \]
\[ Kc = (Vs + (VH/2.0))/\text{pow}(D,3.0); \]

/* CALCULATE Ka AND Kb */
Ka = a/D;
Kb = b/D;

/* CALCULATE THE CYCLONE DESIGN NUMBER */
C = (8.0*Kc)/(Ka*Kb);

/* CALCULATE THE VORTEX PARAMETER FROM ALEXANDER'S EQUATION */
gamma = 1.0-(1.0-(\text{pow}(39.4*D,0.14)/2.5))*\text{pow}((\text{FeedStr->Temp}/283.0),0.3);

Efficiency = 0.0;
for(i=0; i<10; i++)
{
    /* CALCULATE THE INERTIAL IMPACTION PARAMETER FOR EACH SIZE */
    num = \text{PartDens}\cdot\text{pow}(\text{MeanPartSize[i]},2.0)\cdot\text{vel}\cdot(\text{gamma}+1.0);
    den = 18*\text{visc}\cdot D;
    psi = num / den;

    /* CALCULATE THE COLLECTION EFFICIENCY FOR EACH PARTICLE SIZE */
    Eff = 1.0 - \exp(-2.0*\text{pow}((C*\text{psi}),(1.0/(2*\text{gamma}+2))));

    /* CALCULATE THE OVERALL COLLECTION EFFICIENCY */
    Efficiency += Eff*\text{Fraction_In_Range[i]};
}
cycPtr->Eff = Efficiency;

/* MASS BALANCES FOR THE SOLID PARTICLE COMPONENTS */
Copy_Stream(\text{FeedStr},\text{CleanStr});
FeedCompPtr = \text{FeedStr->StCompList};
CleanCompPtr = \text{CleanStr->StCompList};
DustCompPtr = \text{DustStr->StCompList};
for( i=0; i<\text{NumOfComps}; i++)
{
    if(\text{theComps[i]->Density[0]} > 5.0) // Identify solid particles in the feed stream based on density
    {
        AmountRemoved = \text{FeedCompPtr->CompMassFlow}\cdot\text{Efficiency};
        \text{CleanCompPtr->CompMassFlow} *= 1.0 - \text{Efficiency};
        \text{DustCompPtr->CompMassFlow} = \text{AmountRemoved};
    }
    else
    {
        AmountRemoved = 0.0;
        \text{CleanCompPtr->CompMassFlow} = \text{FeedCompPtr->CompMassFlow};
        \text{DustCompPtr->CompMassFlow} = \text{AmountRemoved};
    }

    \text{FeedCompPtr} = \text{FeedCompPtr->next};
    \text{CleanCompPtr} = \text{CleanCompPtr->next};
DustCompPtr = DustCompPtr->next;
}
CleanStr->MassFlow = Estim_MassFlow( CleanStr );
DustStr->MassFlow = Estim_MassFlow( DustStr );
}
else /* Efficiency specified by the user */ {
/* MASS BALANCES FOR THE SOLID PARTICLE COMPONENTS */
Copy_Stream(FeedStr,CleanStr);
FeedCompPtr = FeedStr->StCompList;
CleanCompPtr = CleanStr->StCompList;
DustCompPtr = DustStr->StCompList;
for( i=0; i<NumOfComps; i++) {
    AmountRemoved = FeedCompPtr->CompMassFlow*RemovFrac[i];
    CleanCompPtr->CompMassFlow *= 1.0 - RemovFrac[i];
    DustCompPtr->CompMassFlow = AmountRemoved;
    FeedCompPtr = FeedCompPtr->next;
    CleanCompPtr = CleanCompPtr->next;
    DustCompPtr = DustCompPtr->next;
}
CleanStr->MassFlow = Estim_MassFlow( CleanStr );
DustStr->MassFlow = Estim_MassFlow( DustStr );
}
/* ESTIMATE POWER CONSUMPTION FOR THE BLOWER, kw */
if(!unitPtr->m_isPowerSet) {
    unitPtr->Power = (Q/3600.0)*DP/1000.0;
}
/* ESTIMATE PURCHASE COST */
Cost_Cyclone(unitPtr,Q);
/* ESTIMATE ENVIRONMENTAL PROPERTIES OF EXITING STREAMS */
Estimate_EnviroStreamProperties(DustStr);
Estimate_EnviroStreamProperties(CleanStr);
}
Solve_BagHouseFilter(LINK unitPtr)
{
    BagHouseFilterPtr bhfPtr;
    St_CompPtr CleanCompPtr, DustCompPtr, FeedCompPtr;
    St_LINK FeedStr, DustStr;
    St_LINK CleanStr, shead;
    St_LINK assStream[2];
    short i, StrNum;
    short ncompart; // number of compartments
    double dpm; // maximum pressure drop, Pa
    double MaxVel; // maximum filtering velocity, m/min
    double ctime; // cleaning time, min
    double ftime; // filtration time, min
    double rtime; // run time, min
    double acloth; // net cloth area, m2
    double ke; // filter drag model constant, Pa-min/m
    double ks; // filter drag model constant, Pa-min-m/kg
    double RemovFrac[compMax]; // Removal fraction for each component
    double mflow; // feed mass flow, kg/h
    double Q; // volumetric flow of input gas, m3/min
    double acpcomp; // cloth area in one compartment, m2
    double filvel1; // filtering vel. when one compartment is off-line, m/min
    double filvel2; // filtering vel when every compartment is on-line, m/min
    double vfactor; // factor to find actual filtering velocity
    double acfilvel; // actual filtering vel, m/min
    double fildrag; // filter drag, Pa-min/m
    double dustden; // actual dust density, kg/m2 of fabric
    double dload; // dust loading, kg/m3
    double AmountRemoved; // Amount removed from each component

    bhfPtr = (BagHouseFilterPtr) unitPtr->theModel;

    // FIND FEED AND PRODUCT STREAMS
    shead = stream_head;
    Find_Ass_Stream(unitPtr, assStream, shead, FEED, &StrNum);
    FeedStr = assStream[0];

    shead = stream_head;
    Find_Ass_Stream(unitPtr, assStream, shead, PROD, &StrNum);

    if(EqualPt(assStream[0]->spoints[0], unitPtr->hpoints[2]))
    {
        CleanStr = assStream[0];
        DustStr = assStream[1];
    }
    else
    {
        CleanStr = assStream[1];
        DustStr = assStream[0];
    }

    // GET CURRENT DATA STRUCTURE VALUES
    for(i=0; i<compMax; i++)
RemovFrac[i] = bhfPtr->RemovFrac[i];
MaxVel = bhfPtr->MaxVel;
cTime = bhfPtr->cTime;
ke = bhfPtr->ke;
ks = bhfPtr->ks;
ncompart = bhfPtr->ncompart;

// Check if feed mass flowrate is zero
mflow = FeedStr->MassFlow;
if(fabs(mflow) < 0.0000001)
    return;

// Estimate volumetric flowrate of feed stream, m³/h
Q = Estim_VolFlow(FeedStr)/60.0; // convert m³/h to m³/min

// Estimate net cloth area, m²
acloth = Q/MaxVel;
bhfPtr->acloth = acloth;

// Estimate cloth area per compartment, m²
acpcomp = acloth/ncompart;

// Filtering velocity when one compartment is off-line, m/min
fillvell = Q/((ncompart-1)*acpcomp);

// Filtering velocity when every compartment is on-line, m/min
fillve12 = Q/(ncompart*acpcomp);

// Estimate dust loading, kg/m³
dload = 0.0;
FeedCompPtr = FeedStr->StCompList;
for(i=0; i<NumOfComps; i++)
{
    if( theComps[i]->Density[0] > 1000.0 )
        dload += FeedCompPtr->CompMassFlow;
    FeedCompPtr = FeedCompPtr->next;
}
dload /= Q;

// Correction factor to find actual filtering velocity, Cooper & Alley pp 173
vfactor = exp(0.24488 - 0.41681*log(ncompart) +
               0.058905*pow(log(ncompart),2.0));

// Estimate actual filtering velocity, m/min
acfilvel = vfactor*fillvell;

// Check if user specifies filtration time or maximum pressure drop
if(bhfPtr->FiltrationTimeSpec)// User specifies filtration time in minutes
{
    // Get filtration time from data structure
    ftime = bhfPtr->ftime;
    // Estimate run time, min
    rtime = ((ftime + cTime) / ncompart) - cTime ;
// ESTIMATE ACTUAL DUST DENSITY, kg/m

dustden = (ncompart-1)*dload*((filvel2*rtime) + (filvel1*ctime));

// ESTIMATE FILTER DRAG, Pa-min/m

fildrag = ke + (ks*dustden);

// ESTIMATE MAXIMUM PRESSURE DROP, Pa

dpm = fildrag * acfilvel;

bhfPtr->dpm = dpm;

else // User specifies maximum pressure drop in pascals
{
    // GET MAXIMUM PRESSURE DROP FROM DATA STRUCTURE
    dpm = bhfPtr->dpm;

    // ESTIMATE FILTER DRAG, Pa-min/m
    fildrag = dpm/acfilvel;

    // ESTIMATE ACTUAL DUST DENSITY, KG/m
    dustden = ( fildrag - ke ) / ks;

    // ESTIMATE RUN TIME, min
    rtime = ( (dustden/((ncompart-1)*dload)) - filvell*ctime ) / filvel2 ;

    // ESTIMATE FILTRATION TIME, min
    ftime = ncompart*(rtime + ctime) - ctime ;
    bhfPtr->ftime = ftime;
}

// MASS BALANCES FOR THE SOLID PARTICLE COMPONENTS

Copy_Stream(FeedStr,CleanStr);

FeedCompPtr = FeedStr->StCompList;
CleanCompPtr = CleanStr->StCompList;
DustCompPtr = DustStr->StCompList;

for( i=0; i<NumOfComps; i++) {
    AmountRemoved = CleanCompPtr->CompMassFlow*bhfPtr->RemovFrac[i];
    CleanCompPtr->CompMassFlow *= 1.0 - bhfPtr->RemovFrac[i];
    DustCompPtr->CompMassFlow = AmountRemoved;

    FeedCompPtr = FeedCompPtr->next;
    CleanCompPtr = CleanCompPtr->next;
    DustCompPtr = DustCompPtr->next;
}

CleanStr->MassFlow = Estim_MassFlow( CleanStr );
DustStr->MassFlow = Estim_MassFlow( DustStr );

// ESTIMATE POWER REQUIREMENT, kw
if(!unitPtr->m isPowerSet)
    unitPtr->Power = (Q/60.0)*dpm/1000.0;
// ESTIMATE PURCHASE COST BAG HOUSE FILTER
Cost_BagHouseFilter( unitPtr );

// ESTIMATE ENVIRONMENTAL PROPERTIES OF THE EXIT STREAMS
Estimate_EnviroStreamProperties(DustStr);
Estimate_EnviroStreamProperties(CleanStr);
}

********************************************************************************

* SOLVE ELECTROSTATIC PRECIPITATOR
********************************************************************************

Solve_E_Precipitator(LINK unitPtr)
{
    E_PrecipitatorPtr espPtr;
    St_CompPtr CleanCompPtr, DustCompPtr, FeedCompPtr;
    St_LINK FeedStr, DustStr;
    St_LINK CleanStr, shead;
    St_LINK assStream[2];
    short i, StrNum;
    short StreamPoints;

    double GasVel; /* entering gas velocity, m/min */
    short Nd; /* number of ducts */
    short Ns; /* number of sections */
    double PlatHet; /* plate height, m */
    double PlatLen; /* plate length, m */
    double OvEff; /* overall efficiency */
    double A_Ratio; /* aspect ratio, number */
    double PlatSep; /* plate separation, m */
    short Np; /* number of plates */
    double dp; /* pressure drop, Pa */
    double MeanPartSize[10]; /* mean particle size, micron */
    double Fr_In_Range[10]; /* fraction in size range */
    double MVel_In_Range[10]; /* migration velocity in size range, m/min */
    double mflow; /* feed mass flow, kg/h */
    double Q; /* volumetric flow of input gas, m3/min */
    double A; /* total collection area, m2 */
    double Ap; /* single plate area, m2 */
    double efficiency[10]; /* efficiency for each size range */
    double GasDen; /* Density of entering gas, kg/m3 */
    double AmountRemoved; /* Amount removed from each component */

    double area_left = 0.0; /* Lower limit of Collection Area, m2 */
    double area_right = 1000000.0; /* Upper limit Collection Area, m2 */
    double area_middle; /* Middle point in bisection method */

    double D_left; /* Collection efficiency at area_left, m */
double D_right;  /* Collection efficiency at area_right, m */
double D_middle; /* Collection efficiency at area_middle, m */

espPtr = (E_PrecipitatorPtr) unitPtr->theModel;

/* FIND FEED AND PRODUCT STREAMS */
shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, FEED, &StrNum );
FeedStr = assStream[0];

shead = stream_head;
Find_Ass_Stream( unitPtr, assStream, shead, PROD, &StrNum );

if( EqualPt( assStream[0]->spoints[0], unitPtr->hpoints[2] ) )
{
    CleanStr = assStream[0];
    DustStr = assStream[1];
}
else
{
    CleanStr = assStream[1];
    DustStr = assStream[0];
}

/* GET CURRENT DATA STRUCTURE VALUES */
for(i=0; i<10; i++)
{
    MeanPartSize[i] = espPtr->MeanPartSize[i];
    Fr_In_Range[i] = espPtr->Fr_In_Range[i];
    MVel_In_Range[i] = espPtr->MVel_In_Range[i];
}

GasVel = espPtr->GasVel;
Nd = espPtr->Nd;
Ns = espPtr->Ns;
PlatHet = espPtr->PlatHet;
PlatLen = espPtr->PlatLen;
OvEff = espPtr->OvEff;
A_Ratio = espPtr->A_Ratio;

/* CHECK IF FEED MASS FLOW RATE IS ZERO */
mflow = FeedStr->MassFlow;
if( fabs(mflow) < 0.0000001 )
    return;

/* ESTIMATE VOLUMETRIC FLOW RATE OF FEED STREAM, m3/h */
Q = Estim_VolFlow(FeedStr)/60.0;  /* convert m3/h to m3/min */

/* CHECK IF USER SPECIFIES OVERALL EFFICIENCY OR SINGLE PLATE CHARACTERISTICS */
if(espPtr->UserSpecEff) /* User specifies overall efficiency */
{
    /* APPLY BISECTION METHOD TO CALCULATE THE COLLECTION AREA */
    area_left = -100.0; /* Lower limit of Collection Area, m2 */
    area_right = 1000000.0; /* Upper limit of Collection Area, m2 */
    D_left = Estimate_ESP_Eff(unitPtr,Q,area_left);
    D_right = Estimate_ESP_Eff(unitPtr,Q,area_right);
if((D_left * D_right) < 0.0)
{
    while(fabs(area_left - area_right) > 0.001)
    {
        area_middle = (area_left + area_right)/2.0;
        D_middle = Estimate_ESP_Eff(unitPtr,Q,area_middle);
        if(D_middle*D_left < 0.0)
            area_right = area_middle;
        else
        {
            area_left = area_middle;
            D_left = D_middle;
        }
    }
    A = area_middle;

    /* ESTIMATE SINGLE PLATE LENGTH, m */
    PlatLen = pow( ( (A * A_Ratio) / (2.0 * Nd * pow(Ns,2.0))), 0.5);
    espPtr->PlatLen = PlatLen;

    /* ESTIMATE SINGLE PLATE HEIGHT, m */
    PlatHet = (Ns * PlatLen) / A_Ratio;
    espPtr->PlatHet = PlatHet;

    /* ESTIMATE SINGLE PLATE AREA, m2 */
    Ap = PlatHet * PlatLen;
}
else /* User specifies single plate characteristics */
{
    /* ESTIMATE SINGLE PLATE AREA, m2 */
    Ap = PlatHet * PlatLen;

    /* ESTIMATE TOTAL COLLECTION AREA, m2 */
    A = 2.0 * PlatHet * PlatLen * Nd * Ns;

    /* ESTIMATE EFFICIENCY FOR EACH PARTICLE SIZE RANGE */
    for(i=0; i<10; i++)
        efficiency[i] = 1.0 - exp(( - MVel_In_Range[i] * A ) / Q) ;

    /* ESTIMATE OVERALL EFFICIENCY */
    OvEff = 0.0;
    for(i=0; i<10; i++)
        OvEff += Fr_In_Range[i] * efficiency[i];
    espPtr->OvEff = OvEff;

    /* ESTIMATE ASPECT RATIO */
    espPtr->A_Ratio = (Ns * PlatLen) / PlatHet;
}

/* ESTIMATE PLATE SEPARATION, m */
PlatSep = Q / ( Nd * GasVel * PlatHet ) ;
espPtr->PlatSep = PlatSep;

/* ESTIMATE NUMBER OF PLATES, min */
Np = ( A/Ap ) + 1.0 + 1.0;
espPtr->Np = Np;

/* ESTIMATE DENSITY OF FEED STREAM, kg/m3 */
GasDen = Solvent_Density(FeedStr); /* convert kg/m3 to */

/* ESTIMATE TOTAL PRESSURE DROP, Pa */
dp = 45.5 * GasDen * pow((GasVel/60.0),2.0);
espPtr->dp = dp;

/* ESTIMATE OVERALL COLLECTION AREA, m2 */
A = 2.0 * PlatHet * PlatLen * Nd * Ns;
espPtr->A = A;

/* MASS BALANCES FOR THE SOLID PARTICLE COMPONENTS */

Copy_Stream(FeedStr,CleanStr);

FeedCompPtr = FeedStr->StCompList;
CleanCompPtr = CleanStr->StCompList;
DustCompPtr = DustStr->StCompList;

for( i=0; i<NumOfComps; i++)
{
    if ( theComps[i]->Density[0] > 5.0 ) // Identify solid particles in the
        // feed stream based on density
    {
        AmountRemoved = FeedCompPtr->CompMassFlow*espPtr->OvEff;
        CleanCompPtr->CompMassFlow *= 1.0 - espPtr->OvEff;
        DustCompPtr->CompMassFlow = AmountRemoved;
    }
    else
    {
        AmountRemoved = 0.0;
        CleanCompPtr->CompMassFlow = FeedCompPtr->CompMassFlow;
        DustCompPtr->CompMassFlow = AmountRemoved;
    }

    FeedCompPtr = FeedCompPtr->next;
    CleanCompPtr = CleanCompPtr->next;
    DustCompPtr = DustCompPtr->next;
}

CleanStr->MassFlow = Estim_MassFlow( CleanStr );
DustStr->MassFlow = Estim_MassFlow( DustStr );

/* ESTIMATE POWER REQUIREMENT, kw */
if(!unitPtr->m_isPowerSet)
    unitPtr->Power = (Q/60.0)*dp/1000.0;

/* ESTIMATE PURCHASE COST ELECTROSTATIC PRECIPITATOR */
Cost_E_Precipitator( unitPtr,Q );

/* ESTIMATE ENVIRONMENTAL PROPERTIES OF THE EXIT STREAMS */
double Estimate_ESP_Eff(LINK unitPtr, float Q, float area)
{
  E_PrecipitatorPtr espPtr;

  short i;
  float Fr_In_Range[10]; /* fraction in size range */
  float MVel_In_Range[10]; /* migration velocity in size range, m/min */
  float efficiency[10]; /* efficiency for each size range */
  float CalOvEff; /* Overall calculated efficiency */
  float OvEff; /* overall efficiency supplied by user */
  float diff;

  espPtr = (E_PrecipitatorPtr) unitPtr->theModel;

  /* GET DATA STRUCTURE VALUES */
  for (i = 0; i < 10; i++)
  {
    Fr_In_Range[i] = espPtr->Fr_In_Range[i];
    MVel_In_Range[i] = espPtr->MVel_In_Range[i];
  }

  OvEff = espPtr->OvEff;

  /* ESTIMATE EFFICIENCY FOR EACH PARTICLE SIZE RANGE */
  for (i = 0; i < 10; i++)
  {
    efficiency[i] = 1.0 - exp(( - MVel_In_Range[i] * area ) / Q) ;
  }

  /* ESTIMATE OVERALL EFFICIENCY */
  CalOvEff = 0.0;
  for (i = 0; i < 10; i++)
  {
    CalOvEff += Fr_In_Range[i] * efficiency[i] ;
  }

  /* CALCULATE THE DIFFERENCE BETWEEN CALCULATED OVERALL EFF AND SUPPLIED EFF */
  diff = CalOvEff - OvEff;

  return(diff);
}
APPENDIX III

PROCESS EMISSIONS IN ENVIROCAD
SolveEmissions()
{
LINK unitPtr;
short i, j;

unitPtr = flow_head;
for( i=0; i<NumOfUnits; i++ )
{
    if( unitPtr->Emits )
    {
        for(j=0; j<7; j++)
        {
            if( unitPtr->EmitTasks[j] )
            {
                switch( j )
                {
                    case 0: SolveCharging(unitPtr);
                            break;
                    case 1: SolveEvacuation(unitPtr);
                            break;
                    case 2: SolvePurging(unitPtr);
                            break;
                    case 3: SolveHeating(unitPtr);
                            break;
                    case 4: SolveGasEvolution(unitPtr);
                            break;
                    case 5: SolveVacuum(unitPtr);
                            break;
                    case 6: SolveDrying(unitPtr);
                            break;
                    default: break;
                }
            }
        }
    }
    unitPtr = unitPtr->next;
}

// CREATE EMISSIONS REPORT FILE
doEmissionsReport( );
SolveCharging (LINK unitPtr)
{
    int i; // Dummy index variable
    float Vr; // Rate of air displacement, ft3/hr
    float Lr; // Liquid pumping rate, gpm
    float Temp; // Exit temperature, °C
    float Se[compMax]; // Rate of each VOC emission, lbs/hr
    float sum = 0.0; // Sum of the VOC vapor pressures times their mole fractions
    float TotalSe = 0.0; // Total VOC Emissions from Charging
    float X[compMax]; // Liquid mole fraction for each VOC
    float P[compMax]; // Vapor pressure for each VOC using Antoine's equation
    float a[compMax]; // Antoine's constant
    float b[compMax]; // Antoine's constant
    float c[compMax]; // Antoine's constant
    float MW[compMax]; // MW of each VOC

    // GET CURRENT VALUES FOR THE DATA STRUCTURE
    for(i=0; i < NumOfComps; i++)
    {
        if(theComps[i]->Is_VOC)
        {
            X[i] = theComps[i]->X;
            a[i] = theComps[i]->a;
            b[i] = theComps[i]->b;
            c[i] = theComps[i]->c;
            MW[i] = theComps[i]->MW;
        }
    }

    Temp = unitPtr->EmitTaskParameter[0][1];
    Lr = unitPtr->EmitTaskParameter[0][0];

    // CALCULATE THE VAPOR PRESSURES OF THE VOCs AND THE SUM
    for(i=0; i < NumOfComps; i++)
    {
        if(theComps[i]->Is_VOC)
        {
            P[i] = pow(10.0, a[i] - (b[i]/(c[i]+Temp)));
            sum += P[i]*X[i];
        }
    }

    // CALCULATE THE RATE OF AIR DISPLACEMENT IN ft3/hr
    Vr = Lr * 0.134 * 60.0;

    // CALCULATE THE lbs/hr OF EACH VOC Emitted
    for(i=0; i < NumOfComps; i++)
    {
        if(theComps[i]->Is_VOC)
Se[i] = (P[i] * X[i] * Vr * MW[i]) / (999.0 * (Temp + 273.15));
else
    Se[i] = 0.0;
TotalSe += Se[i];
}

// ASSIGN TOTAL EMISSIONS
unitPtr->EmitTaskParameter[0][2] = TotalSe;

*******************************************************************************
* SolveEvacuation() : Function that solves the Evacuation model
* K.Abeliotis 9/6/1994
*******************************************************************************
SolveEvacuation(LINK unitPtr)
{
    int i;  // Dummy index variable
    float Vi;  // Initial volume of air in the vessel, ft3
    float Vf;  // Final volume of air in the vessel, ft3
    float Vr;  // Rate of air removal from the vessel, ft3/hr
    float Ri;  // Initial ratio of moles air to moles total VOC vapor
    float Rf;  // Final ratio of moles air to moles total VOC vapor
    float Ra;  // Average ratio of moles air to moles total VOC vapor
    float VRS;  // Volume of total VOC vapor discharged, ft3/hr
    float Fs;  // Free space in still, ft3
    float t;  // Time of evacuation, hrs
    float Temp;  // Exit temperature, °C
    float Pal;  // Initial air pressure in the vessel, mmHg
    float Pa2;  // Final air pressure in the vessel, mmHg
    float Se[compMax];  // Rate of each VOC emission, lbs/hr
    float sum = 0.0;  // Sum of the VOC vapor pressures times their mole fractions
    float TotalSe = 0.0;  // Total VOC Emissions from Evacuation
    float X[compMax];  // Liquid mole fraction for each VOC
    float P[compMax];  // Vapor pressure for each VOC using Antoine's equation
    float a[compMax];  // Antoine's constant
    float b[compMax];  // Antoine's constant
    float c[compMax];  // Antoine's constant
    float MW[compMax];  // MW of each VOC

    // GET CURRENT VALUES FOR THE DATA STRUCTURE
    for(i=0; i < NumOfComps; i++)
    {
        if(theComps[i]->Is_VOC)
        {
            X[i] = theComps[i]->X;
            a[i] = theComps[i]->a;
            b[i] = theComps[i]->b;
            c[i] = theComps[i]->c;
            MW[i] = theComps[i]->MW;
        }
    }
    Temp = unitPtr->EmitTaskParameter[1][4];
    Fs = unitPtr->EmitTaskParameter[1][2];
Pal = unitPtr->EmitTaskParameter[1][0];
Pa2 = unitPtr->EmitTaskParameter[1][1];
t = unitPtr->EmitTaskParameter[1][3];

    // CALCULATE THE VAPOR PRESSURES OF THE VOCs AND THE SUM
    for(i=0; i < NumOfComps; i++)
    {
        if(theComps[i]->Is_VOC)
        {
            P[i] = pow(10.0, a[i] - (b[i]/(c[i]+Temp)));
            sum += P[i]*X[i];
        }
    }

    // CALCULATE THE INITIAL VOLUME OF AIR IN THE VESSEL IN ft3 (std)
    Vi = ((Pal - sum)/760.0) * Fs;

    // CALCULATE THE FINAL VOLUME OF AIR IN THE VESSEL IN ft3 (std)
    Vf = ((Pa2 - sum)/760.0) * Fs;

    // CALCULATE THE RATE OF AIR REMOVAL FROM THE VESSEL IN ft3/hr
    Vr = (Vi - Vf)/t;

    // CALCULATE INITIAL RATIO OF AIR TO TOTAL VOC VAPOR
    Ri = (Pal - sum)/sum;

    // CALCULATE FINAL RATIO OF AIR TO TOTAL VOC VAPOR
    Rf = (Pa2 - sum)/sum;

    // CALCULATE AVERAGE RATIO OF AIR TO TOTAL VOC VAPOR
    Ra = (Ri + Rf)/2.0;

    // CALCULATE VOLUME OF TOTAL VOC VAPOR DISCHARGED
    VRS = Vr/Ra;

    // CALCULATE THE lbs/hr OF EACH VOC Emitted
    for(i=0; i < NumOfComps; i++)
    {
        if(theComps[i]->Is_VOC)
        {
            Se[i] = (760.0 * VRS * MW[i])/(999.0 * (Temp + 273.15));
        }
        else
        {
            Se[i] = 0.0;
        }
        TotalSe += Se[i];
    }

    // ASSIGN TOTAL EMISSIONS
    unitPtr->EmitTaskParameter[1][S] = TotalSe;
}

//*****************************************************************************/
/* SolvePurging() : Function that solves the Purging model */
/* K.Abeliotis 9/6/1994 */
//*****************************************************************************/
SolvePurging(LINK unitPtr)
int i; // Dummy index variable
float Ns; // Rate of inert gas sweep, std ft3/min
float Vr1; // Rate of inert gas sweep, std ft3/hr
float Vr2; // Rate of total gas displaced from the vessel, std ft3/hr
float Temp; // Exit temperature, °C
float Se[compMax]; // Rate of each VOC emission, lbs/hr
float sum = 0; // Sum of the VOC vapor pressures times their mole fractions
float TotalSe = 0; // Total VOC Emissions from Charging
float X[compMax]; // Liquid mole fraction for each VOC
float P[compMax]; // Vapor pressure for each VOC using Antoine's equation
float a[compMax]; // Antoine's constant
float b[compMax]; // Antoine's constant
float c[compMax]; // Antoine's constant
float MW[compMax]; // MW of each VOC

// GET CURRENT VALUES FOR THE DATA STRUCTURE
for(i=0; i < NumOfComps; i++)
{
    if(theComps[i]->Is_VOC)
    {
        X[i] = theComps[i]->X;
        a[i] = theComps[i]->a;
        b[i] = theComps[i]->b;
        c[i] = theComps[i]->c;
        MW[i] = theComps[i]->MW;
    }
}

Temp = unitPtr->EmitTaskParameter[2][1];
Ns = unitPtr->EmitTaskParameter[2][0];

// CALCULATE THE VAPOR PRESSURES OF THE VOCs AND THE SUM
for(i=0; i < NumOfComps; i++)
{
    if(theComps[i]->Is_VOC)
    {
        P[i] = pow(10.0, a[i] - (c[i]+TeTemp));
        sum += P[i]*X[i];
    }
}

// CALCULATE THE RATE OF INERT GAS SWEEP IN ACTUAL ft3/hr (AT EXIT TEMPERATURE
Vr1 = Ns + 60 * ((273.15 + Temp) / 298.15); // CALCULATE THE RATE OF TOTAL GAS DISPLACED FROM THE VESSEL IN ft3/hr
Vr2 = Vr1*(760.0 / (760.0 - sum));

// CALCULATE THE lbs/hr OF EACH VOC EMMITED
for(i=0; i < NumOfComps; i++)
{
    if(theComps[i]->Is_VOC)
        Se[i] = (P[i] * X[i] * Vr2 * MW[i])/(999.0 * (Temp + 273.15));
    else
        Se[i] = 0.0;
TotalSe += Se[i];
}

// ASSIGN TOTAL EMISSIONS
unitPtr->EmitTaskParameter[2][2] = TotalSe;
}

/**********************
 * SolveHeating() : Function that solves the Heating model
 * K.Abeliotis 9/6/1994
 */

SolveHeating(LINK unitPtr)
{
    int i;       // Dummy index variable
    float T1;   // Initial Temperature of still, °C
    float T2;   // Final Temperature of still, °C
    float Tr;   // Receiver Temperature, °C
    float V;    // Volume of free space in still, ft3
    float dn;   // Number of lbmoles of air displaced to the receiver
    float ns;   // Number of lbmoles of VOC vapor displaced from the receiver
    float Pal;  // Initial air pressure in the vessel, mmHg
    float Pa2;  // Final air pressure in the vessel, mmHg
    float sum = 0.0; // Sum of the VOC vapor pressures times their mole fractions
    float TotalSe = 0.0; // Total VOC Emissions from Evacuation
    float X[compMax]; // Liquid mole fraction for each VOC
    float P[compMax]; // Vapor pressure for each VOC using Antoine's equation
    float a[compMax]; // Antoine's constant
    float b[compMax]; // Antoine's constant
    float c[compMax]; // Antoine's constant
    float MW[compMax]; // MW of each VOC

    // GET CURRENT VALUES FOR THE DATA STRUCTURE
    for(i=0; i < NumOfComps; i++)
    {
        if(theComps[i]->Is_VOC)
        {
            X[i] = theComps[i]->X;
            a[i] = theComps[i]->a;
            b[i] = theComps[i]->b;
            c[i] = theComps[i]->c;
            MW[i] = theComps[i]->MW;
        }
    }

    T1 = unitPtr->EmitTaskParameter[3][0];
    T2 = unitPtr->EmitTaskParameter[3][1];
    Tr = unitPtr->EmitTaskParameter[3][3];
    V = unitPtr->EmitTaskParameter[3][2];

    // CALCULATE THE VAPOR PRESSURES OF THE VOCs AND THE SUM AT THE INITIAL TEMP.
    for(i=0; i < NumOfComps; i++)
    {
if(theComps[i]->Is_VOC)
{
    P[i] = pow(10.0, a[i] - (b[i]/(c[i]+T1)));
    sum += P[i]*X[i];
}

// CALCULATE THE INITIAL PRESSURE OF AIR IN THE STILL IN mmHg
Pal = 760.0 - sum;

// CALCULATE THE VAPOR Pressures OF THE VOCs AND THE SUM AT THE FINAL TEMP.
sum = 0;
for(i=0; i < NumOfComps; i++)
{
    if(theComps[i]->Is_VOC)
    {
        P[i] = pow(10.0, a[i] - (b[i]/(c[i]+T2)));
        sum = sum + (P[i]*X[i]);
    }
}

// CALCULATE THE FINAL PRESSURE OF AIR IN THE STILL IN mmHg
Pa2 = 760.0 - sum;
if(Pa2 < 0.0)    // Check if final air pressure is negative
    Pa2 = 0.0;

// CALCULATE THE LB-MOLES OF AIR DISPLACED TO THE RECEIVER
dn = (V/999.0)*((Pal/(273.15 + Ti)) - (Pa2/(273.15 + T2)));

// CALCULATE THE VAPOR Pressures OF THE VOCs AND THE SUM AT THE RECEIVER TEMP.
sum = 0;
for(i=0; i < NumOfComps; i++)
{
    if(theComps[i]->Is_VOC)
    {
        P[i] = pow(10.0, a[i] - (b[i]/(c[i]+Tr)));
        sum += P[i]*X[i];
    }
}

// CALCULATE THE LB-MOLES OF VOC VAPOR DISPLACED FROM THE RECEIVER
ns = (sum/(760.0 - sum)) * dn;

// CALCULATE THE lbs/hr OF EACH VOC Emitted
for(i=0; i < NumOfComps; i++)
{
    if(theComps[i]->Is_VOC)
        Se[i] = ns * MW[i] *X[i];
    else
        Se[i] = 0.0;
    TotalSe += Se[i];
}

// ASSIGN TOTAL EMISSIONS
unitPtr->EmitTaskParameter[3][4] = TotalSe;
```c
SolveGasEvolution(LINK unitPtr)
{
    int i;  // Dummy index variable
    float Pv;  // Pressure in the vessel, atm
    float Wg;  // Rate of gas evolution, lbs/hr
    float MWg;  // Molecular weight of gas evolved, lbs/lbmole
    float Vrl;  // Rate of gas evolution, ft3/hr
    float Vr2;  // Rate of gas displacement, ft3/hr
    float Temp;  // Exit temperature, °C
    float Se[compMax];  // Rate of each VOC emission, lbs/hr
    float sum = 0;  // Sum of the VOC vapor pressures times their mole fractions
    float TotalSe = 0;  // Total VOC Emissions from Charging
    float X[compMax];  // Liquid mole fraction for each VOC
    float P[compMax];  // Vapor pressure for each VOC using Antoine's equation
    float a[compMax];  // Antoine's constant
    float b[compMax];  // Antoine's constant
    float c[compMax];  // Antoine's constant
    float MW[compMax];  // MW of each VOC

    // GET CURRENT VALUES FOR THE DATA STRUCTURE
    for(i=0; i < NumOfComps; i++)
    {
        if(theComps[i]->Is_VOC)
        {
            X[i] = theComps[i]->X;
            a[i] = theComps[i]->a;
            b[i] = theComps[i]->b;
            c[i] = theComps[i]->c;
            MW[i]= theComps[i]->MW;
        }
    }

    Temp = unitPtr->EmitTaskParameter[4][3];
    Pv = unitPtr->EmitTaskParameter[4][0];
    Wg = unitPtr->EmitTaskParameter[4][1];
    MWg = unitPtr->EmitTaskParameter[4][2];

    // CALCULATE THE VAPOR PRESSURES OF THE VOCs AND THE SUM
    for(i=0; i < NumOfComps; i++)
    {
        if(theComps[i]->Is_VOC)
        {
            P[i] = pow(10.0, a[i] - (b[i]/(c[i]+Temp)));
            sum += P[i]*X[i];
        }
    }

    // CALCULATE THE RATE OF GAS EVOLUTION IN ft3/hr
    Vrl = (Wg * 1.314 * (Temp + 273.15)) / (Pv * MWg);
```
// CALCULATE THE RATE OF GAS DISPLACEMENT IN ft3/hr
Vr2 = Vr1 * (760.0/(760.0 - sum));

// CALCULATE THE lbs/hr OF EACH VOC Emitted
for(i=0; i < NumOfComps; i++)
{
    if(theComps[i]->Is_VOC)
        Se[i] = (P[i] * X[i] * Vr2 * MW[i])/(999.0 * (Temp + 273.15));
    else
        Se[i] = 0.0;
    TotalSe += Se[i];
}

// ASSIGN TOTAL EMISSIONS
unitPtr->EmitTaskParameter[4][4] = TotalSe;
}

/**************************************************************************
* SolveVacuum() : Function that solves the Vacuum model
* K.Abeliotis 9/6/1994
**************************************************************************/
SolveVacuum(LINK unitPtr)
{
    int i; // Dummy index variable
    float La; // Leak rate of air into the system, lbs/hr
    float Vrl; // Leak rate of air into the system, ft3/hr
    float Tstill; // Temperature of the still, °C
    float Tsystem; // Temperature of the receiver, °C
    float Psystem; // Absolute Pressure of the receiver, mmHg
    float P1; // Absolute Pressure at start of the test, mmHg
    float P2; // Absolute Pressure at end of test, mmHg
    float Fs; // Total free space under vacuum, ft3
    float t; // Time of test, hrs
    float Se[compMax]; // Rate of each VOC emission, lbs/hr
    float sum = 0.0; // Sum of the VOC vapor pressures times their mole fractions
    float TotalSe = 0.0; // Total VOC Emissions from Evacuation
    float X[compMax]; // Liquid mole fraction for each VOC
    float P[compMax]; // Vapor pressure for each VOC using Antoine's equation
    float a[compMax]; // Antoine's constant
    float b[compMax]; // Antoine's constant
    float c[compMax]; // Antoine's constant
    float MW[compMax]; // MW of each VOC

    // GET CURRENT VALUES FOR THE DATA STRUCTURE
    for(i=0; i < NumOfComps; i++)
    {
        if(theComps[i]->Is_VOC)
            {
                X[i] = theComps[i]->X;
                a[i] = theComps[i]->a;
                b[i] = theComps[i]->b;
                c[i] = theComps[i]->c;
                MW[i] = theComps[i]->MW;
            }
    }

La = unitPtr->EmitTaskParameter[5][0];
Tstill = unitPtr->EmitTaskParameter[5][1];
Tsystem = unitPtr->EmitTaskParameter[5][2];
Psystem = unitPtr->EmitTaskParameter[5][3];
Fs = unitPtr->EmitTaskParameter[5][6];
P1 = unitPtr->EmitTaskParameter[5][4];
P2 = unitPtr->EmitTaskParameter[5][5];
Fs = unitPtr->EmitTaskParameter[5][6];
t = unitPtr->EmitTaskParameter[5][7];

// CALCULATE THE VAPOR PRESSURES OF THE VOCs AND THE SUM
for (i=0; i < NumOfComps; i++)
{
    if (theComps[i]->Is_VOC)
    {
        P[i] = pow(10.0, a[i] - (b[i]/(c[i]+Tsystem)));
        sum += P[i]*X[i];
    }
}

if (fabs(unitPtr->EmitTaskParameter[5][8] - 1.0) < 1.0e-8 )
{
    // CALCULATE THE lbs/hr OF EACH VOC EMITTED
    for(i=0; i < NumOfComps; i++)
    {
        if (theComps[i]->Is_VOC)
        {
            Se[i] = ((La * MW[i])/29.0)*((Psystem/ (Psystem-P[i])) - 1.0)
        }
    }
    TotalSe += Se[i];
}
else // LEAK RATE SPECIFIED BY EXPERIMENT
{
    Vr1 = ((273.0*Fs)/((Tstill + 273.15)*t))*((P2-P1)/760.0);

    // CALCULATE THE lbs/hr OF EACH VOC EMITTED
    for(i=0; i < NumOfComps; i++)
    {
        if (theComps[i]->Is_VOC)
        {
            Se[i] = ((Vr1*MW[i])/359.0)*((Psystem/ (Psystem-P[i]))-1.0);}
        else
        {
            Se[i] = 0.0;
        }
    }
    TotalSe += Se[i];
}

// ASSIGN TOTAL EMISSIONS
unitPtr->EmitTaskParameter[5][9] = TotalSe;
SolveDrying(LINK unitPtr) {
    int i;  // Dummy index variable
    float BW;  // Weight of dry batch, lbs
    float t;  // Time of drying operation, hrs
    float PS1;  // Percent of VOC in wet material into dryer
    float PS2;  // Percent of VOC in less wet material from dryer
    float Se[compMax];  // Rate of each VOC emission, lbs/hr
    float TotalSe = 0.0;  // Total VOC Emissions from Charging

    BW = unitPtr->EmitTaskParameter[6][0];
    t = unitPtr->EmitTaskParameter[6][1];
    PS1 = unitPtr->EmitTaskParameter[6][2];
    PS2 = unitPtr->EmitTaskParameter[6][3];

    // CALCULATE THE RATE OF VOC EMISSIONS, lbs/hr
    TotalSe = (BW/t)*((PS1/(100.0 - PS1)) - (PS2/(100.0 - PS2)));

    // ASSIGN TOTAL EMISSIONS
    unitPtr->EmitTaskParameter[6][4] = TotalSe;
}

doEmissionsReport() {
    FILE *fp;
    LINK unitPtr;
    SFReply theReply;
    Str255 fileName, windName;
    short theCount, i;
    char MyFile[64];
    OSErr theError = noErr;

    if( flow_head == NULL )
        return;

    GetWTitle(flowWindow, windName);
    myPtcstr( windName, MyFile );
    theCount = strlen( MyFile );
    i = 0;
    while( MyFile[i] != '.' && i < theCount ) {
        i++;
        MyFile[i] = 0;
        if( compares(MyFile, fileName) ) {
            theError = myCreateFile(fp, fileName);
            if( theError != noErr )
                return;
            break;
        }
    }

MyFile[i]  = ' . ';  
MyFile[i+1] = 'e';  
MyFile[i+2] = 'm';  
MyFile[i+3] = '\0';  

myCtoPstr( MyFile, fileName );

if( !CreateSaveFile( &theReply, fileName, "pSave Emissions Report As..." ))
    return;

SetCursor( &delayCursor );

myPtoCstr( theReply.fName, MyFile );

if( (fp = fopen(MyFile, "w")) == NULL ) {
    /* postErr(NULL); */
    return;
}

GenerateEmissionsReport(fp);

if( fclose(fp) )
    return;  /* fclose returns 0 when succeeds */

****************************************************************************
*     GenerateEmissionsReport: Function that generates the actual report
*     K.Abeliotis 9/6/1994
*     ************************************************************************
GenerateEmissionsReport( FILE *fp )
{
    LINK unitPtr;
    short i,j;

    unitPtr = flow_head;

    fprintf( fp, "OVERALL PROCESS EMISSIONS PER UNIT OPERATION \n" );
    fprintf( fp, "="*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'\n" );
    fprintf( fp, "="*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'\n" );
    fprintf( fp, "="*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'\n" );

while( unitPtr != NULL )
{
    if( unitPtr->Emits )
    {
        fprintf( fp, "%s\t\t\t" ,unitPtr->name);
        for(j=0; j<7; j++)
        {
if ( unitPtr->EmitTasks[j] )
{
    switch ( j )
    {
    case 0:
        fprintf ( fp, "CHARGING \t\t\t\t" );
        fprintf ( fp, "%6.2f\n\t\t\t\t", unitPtr->EmitTaskParameter[0][2] );
        break;
    case 1:
        fprintf ( fp, "EVACUATION \t\t\t\t" );
        fprintf ( fp, "%6.2f\n\t\t\t\t", unitPtr->EmitTaskParameter[1][5] );
        break;
    case 2:
        fprintf ( fp, "PURGING \t\t\t\t" );
        fprintf ( fp, "%6.2f\n\t\t\t\t", unitPtr->EmitTaskParameter[2][2] );
        break;
    case 3:
        fprintf ( fp, "HEATING \t\t\t\t" );
        fprintf ( fp, "%6.2f\n\t\t\t\t", unitPtr->EmitTaskParameter[3][4] );
        break;
    case 4:
        fprintf ( fp, "EVOLUTION \t\t\t\t" );
        fprintf ( fp, "%6.2f\n\t\t\t\t", unitPtr->EmitTaskParameter[4][4] );
        break;
    case 5:
        fprintf ( fp, "VACUA \t\t\t\t\t" );
        fprintf ( fp, "%6.2f\n\t\t\t\t", unitPtr->EmitTaskParameter[5][9] );
        break;
    case 6:
        fprintf ( fp, "DRYING \t\t\t\t\t" );
        fprintf ( fp, "%6.2f\n\t\t\t\t", unitPtr->EmitTaskParameter[6][4] );
        break;
    default:
    break;
    }
}
fprintf ( fp, "\n" );
}
unitPtr = unitPtr->next;
APPENDIX IV

LIFE CYCLE INVENTORY ANALYSIS IN ENVIROCAD
// Initializes the Life Cycle Analysis data structure
// Kostas 5/31/95

int Init_LifeCycleAnalysis(void)
{
    int i;

    LCAPtr = (LifeCycleAnalysisPtr) NewPtr(sizeof(LIFECYCLE));

    // GET FIRST ELECTRICITY ECO-VECTOR
    // LCAPtr->m_isElectricityEcovector = TRUE;

    for(i=0; i<2; i++)
    {
        LCAPtr->m_EnergyCoal[i] = 0.0;
        LCAPtr->m_EnergyNaturalGas[i] = 0.0;
        LCAPtr->m_EnergyOil[i] = 0.0;
        LCAPtr->m_EnergyCO2[i] = 0.0;
        LCAPtr->m_EnergyNOx[i] = 0.0;
        LCAPtr->m_EnergyVOC[i] = 0.0;
        LCAPtr->m_EnergyCO[i] = 0.0;
        LCAPtr->m_EnergyDust[i] = 0.0;
        LCAPtr->m_EnergyBOD5[i] = 0.0;
        LCAPtr->m_EnergyCOD[i] = 0.0;
        LCAPtr->m_EnergyTSS[i] = 0.0;
        LCAPtr->m_EnergyTDS[i] = 0.0;
        LCAPtr->m_EnergySolids[i] = 0.0;
    }

    return 0;
}

/* ************************************************** */
/* Identify Product Streams For Life Cycle Analysis */
/* K. Abeliotis 8/19/1994 */
/* ************************************************** */
IdentifyLCAProductStreams()
{
    DialogPtr LCADlg; short itemHit;
    short i,j;
    char str1[16], str2[16];

    ListHandle myList; int ListItem; int EditItem; int ColNum;
    int row, col, len, counter;
    Cell curCell;

    St_LINK shead = stream_head;

    // MAKE SURE THAT A FLOWSHET EXISTS
    if(shead == NULL)
{ BioAlert(BIO_ALRT_OK, "A flowsheet is needed for application of LCA."); return; }

// DO THE DRAWING OF THE DIALOG WINDOW
LCADlg = GetNewDialog( 1100, OL, (WindowPtr)(-1L) );
SetCursor( &arrow );
SetPort( LCADlg );
ShowWindow( (WindowPtr)LCADlg );
DrawDialog( LCADlg );
OK_and_Gray_Line(LCADlg, OK, FALSE, 0);

ListItem = 3;
EditItem = 4;
ColNum = 2;

myList = (ListHandle)MakeListDialog(LCADlg, ListItem, EditItem, ColNum);

// INPUT DATA ABOUT STREAMS TO myList
counter = 0;
for(i=0; i<NumOfStreams; i++)
{
    if( !strcmp(shead->to_unit, "PRODUCT") )
    {
        len = strlen(shead->name);
        if( len > 0 )
        {
            row = LAddRow(1, 999, myList);
            col = 0;
            SetPt( &curCell, col, counter );
            
            sprintf( str1, "%s", shead->name);
            LSetCell( &str1, len, curCell, myList);
            curCell.h += 1;
            if( shead->m_LCAProduct )
                sprintf( str1, "%s", "Product");
            else
                sprintf( str1, "%s", "Waste");
            len = strlen( str1 );
            LSetCell( &str1, len, curCell, myList );

            counter++;
        }
    }
    shead = shead->next;
}

SetPt( &curCell, 0, 0);
EnterCell( LCADlg, myList, curCell, EditItem );
PutCurCell( myList, curCell );
dlgStart:
   do {
      ModalDialog( (ProcPtr)filterProc, &itemHit);
   } while( itemHit > Cancel );

shead = stream_head;

if( itemHit == OK )
{
   // GET NEW VALUES
   counter = 0;
   for( i=0; i<NumOfStreams; i++ )
   {
      if( !strcmp(shead->to_unit, "PRODUCT") )
      {
         row = counter;
         col = 1;
         SetPt( &curCell, col, row );

         len = CellExtract( curCell, myList, str1 );
         if( str1[0] == 'P' || str1[0] == 'p' )
            shead->m_LCAProduct = TRUE;
         else
            shead->m_LCAProduct = FALSE;

         counter++;
      }
      shead = shead->next;
   }

   KillList( myList );
   DisposDialog( LCADlg );
   return( itemHit );
}

Data_LCAFeedStreams()
{
   int i;
   short theReturn;
   St_LINK shead = stream_head;

   for(i=0; i<NumOfStreams; i++)
   {
      if( !strcmp(shead->from_unit, "FEED") )
      {
         theReturn = InitializeMassEcovectors(shead);
      }
   }
}
if( theReturn != OK )
    return( theReturn );

    shead = shead->next;
}
return( theReturn );

Data_LCAWasteStreams()
{
    int i;
    short    theReturn;
    St_LINK    shead = stream_head;

    for(i=0; i<NumOfStreams; i++)
    {
        if( !strcmp(shead->to_unit, "PRODUCT") && !shead->m_LCAProduct)

            theReturn = InitializeMassEcovectors(shead);

            if( theReturn != OK )
                return( theReturn );
        }
        shead = shead->next;
    }
return( theReturn );

}

InitializeMassEcovectors(St_LINK shead)
{
    DialogPtr    LCADlg;
    short    itemHit;
    short    i,j;
    Str255    title;
    char    str1[16], str2[16], str3[16], str4[16], windowTitle[32];
    Boolean  m_isOutputStream;

    // DO THE DRAWING OF THE DIALOG WINDOW
    LCADlg = GetNewDialog( 1101, OL, (WindowPtr)(-1L));
    SetCursor( &arrow );
    SetPort( LCADlg );
    OK_and_Gray_Line(LCADlg, OK, FALSE, 0);

    // SET WINDOW TITLE
    strcpy( windowTitle, shead->name );
    strcat( windowTitle, " Mass Eco-vector ");
myCtoPstr( windowTitle, title );
SetWTitle( LCADlg, title );

// INPUT DATA ABOUT COMPONENTS TO myList
sprintf( strl, "% .2e", shead->m_MassCoal );
sprintf( str2, "% .2e", shead->m_MassNaturalGas );
sprintf( str3, "% .2e", shead->m_MassOil );
SetDText( LCADlg, 3, strl );
SetDText( LCADlg, 4, str2 );
SetDText( LCADlg, 5, str3 );

sprintf( strl, "% .2e", shead->mMassCO2 );
sprintf( str2, "% .2e", shead->m_MassNOx );
sprintf( str3, "% .2e", shead->m_MassVOC );
sprintf( str4, "% .2e", shead->m_MassCO );
SetDText( LCADlg, 6, strl );
SetDText( LCADlg, 7, str2 );
SetDText( LCADlg, 8, str3 );
SetDText( LCADlg, 9, str4 );

sprintf( strl, "% .2e", shead->m_MassDust );
sprintf( str2, "% .2e", shead->m_MassBOD5 );
sprintf( str3, "% .2e", shead->m_MassCOD );
sprintf( str4, "% .2e", shead->m_MassTSS );
SetDText( LCADlg, 10, strl );
SetDText( LCADlg, 11, str2 );
SetDText( LCADlg, 12, str3 );
SetDText( LCADlg, 13, str4 );

sprintf( strl, "% .2e", shead->m_MassTDS );
sprintf( str2, "% .2e", shead->m_MassSolids );
SetDText( LCADlg, 14, strl );
SetDText( LCADlg, 15, str2 );

// CHECK IF TARGET STREAM IS AN OUTPUT STREAM IN GENERAL
if( !strcmp(shead->from_unit, "FEED") )
    m_isOutputStream = FALSE;
else
    m_isOutputStream = TRUE;

// CHECK IF TARGET STREAM IS AN INTERMEDIATE STREAM
CheckForIntermediateStream(shead);
    if((strcmp(shead->from_unit, "FEED") ) && (strcmp(shead->to_unit, "PRODUCT")) )
        // Intermediate Stream
        shead->m_LCAProduct = TRUE;

// SET ALL BUTTONS TO FALSE
SetDCtl( LCADlg, 16, FALSE );
SetDCtl( LCADlg, 17, FALSE );
SetDCtl( LCADlg, 18, FALSE );

if(!m_isOutputStream)
{
    SetDCtl( LCADlg, 16, TRUE );
    SetDCtl( LCADlg, 17, FALSE );
    SetDCtl( LCADlg, 18, FALSE );
}
else if(m_isOutputStream && shead->m_LCAProduct)
{
    SetDct( LCADlg, 16, FALSE );
    SetDct( LCADlg, 17, TRUE );
    SetDct( LCADlg, 18, FALSE );
}
else
{
    SetDct( LCADlg, 16, FALSE );
    SetDct( LCADlg, 17, FALSE );
    SetDct( LCADlg, 18, TRUE );
}

dlgStart:
do {
    ModalDialog(OL, &itemHit);
    if(m_isOutputStream)
    {
        if(!strcmp(shead->to_unit, "PRODUCT")) // Only if Product Strea
            // allow user to play
        {
            switch(itemHit)
            {
            case 17:
                SetDct( LCADlg, 16, FALSE );
                SetDct( LCADlg, 17, TRUE );
                SetDct( LCADlg, 18, FALSE );
                shead->m_LCAProduct = TRUE;
                break;
            case 18:
                SetDct( LCADlg, 16, FALSE );
                SetDct( LCADlg, 17, FALSE );
                SetDct( LCADlg, 18, TRUE );
                shead->m_LCAProduct = FALSE;
                break;
            default:
                break;
            }
        }
    }
} while(itemHit > Cancel );

if( itemHit == OK )
{
    // GET NEW VALUES
    GetDText( LCADlg, 3, str1 );
    GetDText( LCADlg, 4, str2 );
    GetDText( LCADlg, 5, str3 );
sscanf( str1, "%e", &shead->m_MassCoal );
sscanf( str2, "%e", &shead->m_MassNaturalGas );
sscanf( str3, "%e", &shead->m_MassOil );

GetDText( LCADlg, 6, str1 );
GetDText( LCADlg, 7, str2 );
GetDText( LCADlg, 8, str3 );
GetDText( LCADlg, 9, str4 );
sscanf( str1, "%e", &shead->m_MassCO2 );
sscanf( str2, "%e", &shead->m_MassNOx );
sscanf( str3, "%e", &shead->m_MassVOC );
sscanf( str4, "%e", &shead->m_MassCO );

GetDText( LCADlg, 10, str1 );
GetDText( LCADlg, 11, str2 );
GetDText( LCADlg, 12, str3 );
GetDText( LCADlg, 13, str4 );
sscanf( str1, "%e", &shead->m_MassDust );
sscanf( str2, "%e", &shead->m_MassBOD5 );
sscanf( str3, "%e", &shead->m_MassCOD );
sscanf( str4, "%e", &shead->m_MassTSS );

GetDText( LCADlg, 14, str1 );
GetDText( LCADlg, 15, str2 );
sscanf( str1, "%e", &shead->m_MassTDS );
sscanf( str2, "%e", &shead->m_MassSolids );

}

DisposeDialog( LCADlg );
return( itemHit );

}

CheckForIntermediateStream(St_LINK shead)
{
    if ( (strcmp(shead->from_unit, "FEED")) && (strcmp(shead->to_unit, "PRODUCT")))
        shead->m_LCAProduct = TRUE;
}

Data_LCAEnergyEcovectors()
{
    short theReturn;

    // GET ELECTRICITY m_isElectricityEcovector = 1
    theReturn = Get_LCAEnergyEcovectors(LCAPtr->m_isElectricityEcovector);

    if( theReturn == OK )
        // GET STEAM m_isElectricityEcovector = 0
        theReturn = Get_LCAEnergyEcovectors(!LCAPtr->m_isElectricityEcovector);

    return( theReturn );
}
Get_LCAEnergyEcoVectors(short i)
{
    DialogPtr    LCADlg;
    short        itemHit;
    Str255       title;
    char         str1[16], str2[16], str3[16], str4[16], windowTitle[32];

    // DO THE DRAWING OF THE DIALOG WINDOW
    LCADlg = GetNewDialog( 1102, 0L, (WindowPtr)(-1L));
    SetCursor( &arrow );
    SetPort( LCADlg );
    OK_and_Gray_Line(LCADlg, OK, FALSE, 0);

    // SET WINDOW TITLE
    if(i == 1)
    {
        strcpy( windowTitle, " Electricity Eco-vector ");
    }
    else
    {
        strcpy( windowTitle, " Steam Eco-vector ");
    }

    myCtoPstr( windowTitle, title );
    SetWTitle( LCADlg, title );

    // INPUT DATA ABOUT COMPONENTS TO myList
    sprintf( str1, "%2e", LCAPtr->m_EnergyCoal[i] );
    sprintf( str2, "%2e", LCAPtr->m_EnergyNaturalGas[i] );
    sprintf( str3, "%2e", LCAPtr->m_EnergyOil[i] );
    SetDText( LCADlg, 3, str1 );
    SetDText( LCADlg, 4, str2 );
    SetDText( LCADlg, 5, str3 );

    sprintf( str1, "%2e", LCAPtr->m_EnergyCO2[i] );
    sprintf( str2, "%2e", LCAPtr->m_EnergyNOx[i] );
    sprintf( str3, "%2e", LCAPtr->m_EnergyVOC[i] );
    sprintf( str4, "%2e", LCAPtr->m_EnergyCO[i] );
    SetDText( LCADlg, 6, str1 );
    SetDText( LCADlg, 7, str2 );
    SetDText( LCADlg, 8, str3 );
    SetDText( LCADlg, 9, str4 );

    sprintf( str1, "%2e", LCAPtr->m_EnergyDust[i] );
    sprintf( str2, "%2e", LCAPtr->m_EnergyBOD5[i] );
    sprintf( str3, "%2e", LCAPtr->m_EnergyCOD[i] );
    sprintf( str4, "%2e", LCAPtr->m_EnergyTSS[i] );
    SetDText( LCADlg, 10, str1 );
    SetDText( LCADlg, 11, str2 );
    SetDText( LCADlg, 12, str3 );
    SetDText( LCADlg, 13, str4 );

    sprintf( str1, "%2e", LCAPtr->m_EnergyTDS[i] );
    sprintf( str2, "%2e", LCAPtr->m_EnergySolids[i] );
    SetDText( LCADlg, 14, str1 );
    SetDText( LCADlg, 15, str2 );

    dlgStart:
    do{
        ModalDialog(OL, &itemHit);
    }
while( itemHit > Cancel );

if( itemHit == OK )
{
    // GET NEW VALUES
    GetDText( LCADlg, 3, str1 );
    GetDText( LCADlg, 4, str2 );
    GetDText( LCADlg, 5, str3 );
    sscanf( str1, "%e", &LCAPtr->m_EnergyCoal[i] );
    sscanf( str2, "%e", &LCAPtr->m_EnergyNaturalGas[i] );
    sscanf( str3, "%e", &LCAPtr->m_EnergyOil[i] );
    GetDText( LCADlg, 6, str1 );
    GetDText( LCADlg, 7, str2 );
    GetDText( LCADlg, 8, str3 );
    GetDText( LCADlg, 9, str4 );
    sscanf( str1, "%e", &LCAPtr->m_EnergyCO2[i] );
    sscanf( str2, "%e", &LCAPtr->m_EnergyNOx[i] );
    sscanf( str3, "%e", &LCAPtr->m_EnergyVOC[i] );
    sscanf( str4, "%e", &LCAPtr->m_EnergyCO[i] );
    GetDText( LCADlg, 10, str1 );
    GetDText( LCADlg, 11, str2 );
    GetDText( LCADlg, 12, str3 );
    GetDText( LCADlg, 13, str4 );
    sscanf( str1, "%e", &LCAPtr->m_EnergyDust[i] );
    sscanf( str2, "%e", &LCAPtr->m_EnergyBOD5[i] );
    sscanf( str3, "%e", &LCAPtr->m_EnergyCOD[i] );
    sscanf( str4, "%e", &LCAPtr->m_EnergyTSS[i] );
    GetDText( LCADlg, 14, str1 );
    GetDText( LCADlg, 15, str2 );
    sscanf( str1, "%e", &LCAPtr->m_EnergyTDS[i] );
    sscanf( str2, "%e", &LCAPtr->m_EnergySolids[i] );
}

DisposDialog( LCADlg );
return( itemHit );

DoLifeCycleAnalysis()
{
    LINK unitPtr;
    St_LINK shead;
    St_LINK assProdStr[NUM_OF_MAX_FEED_STREAMS+2];
    short i, ProdStrNum;
    Boolean m_atLeastOneProductExists = FALSE;

    unitPtr = flow_head;
}
shead = stream_head;

// MAKE SURE THAT A FLOWSHEET EXISTS
if(shead == NULL) {
    BioAlert(BIO_ALRT_OK, "A flowsheet is needed for application of LCA.");
    return;
}

// MAKE SURE THAT THE USER HAS SPECIFIED AT LEAST ONE LCA PRODUCT STREAM
// AND THIS STREAM IS NOT AN INTERMEDIATE STREAM
for(i=0; i<NumOfStreams; i++) {
    if(shead->m_LCAProduct &&
       !(strcmp(shead->from_unit, "FEED") &&
        strcmp(shead->to_unit, "PRODUCT")) {            
        m_atLeastOneProductExists = TRUE;
        break;
    }
    shead = shead->next;
}

if(!m_atLeastOneProductExists) {
    BioAlert(BIO_ALRT_OK, "Please specify at least one output stream as 'product' for the purposes of LCA.");
    return;
}

while( unitPtr != NULL) {
    LCA_UnitOp(unitPtr);
    unitPtr = unitPtr->next;
}

LCA_UnitOp(LINK unitPtr)
{

    St_LINK shead;
    St_LINK assFeedStr[NUM_OF_MAX_FEED_STREAMS+2];
    St_LINK assProdStr[NUM_OF_MAX_FEED_STREAMS+2];
    short i, j, FeedStrNum, ProdStrNum;
    double m_splitMassFraction[NUM_OF_MAX_FEED_STREAMS+2];
    double m_totalProductMass;
    double m_FeedMassCoal = 0.0;        // LCA Variables - EL of Feed Stream
    double m_FeedMassNaturalGas = 0.0;
    double m_FeedMassOil = 0.0;
    double m_FeedMassCO2 = 0.0;
    double m_FeedMassNOx = 0.0;
    double m_FeedMassVOC = 0.0;
// LCA Variables - EL of Waste Streams
double m_WasteMassCoal = 0.0;
double m_WasteMassNaturalGas = 0.0;
double m_WasteMassOil = 0.0;
double m_WasteMassCO2 = 0.0;
double m_WasteMassNOx = 0.0;
double m_WasteMassVOC = 0.0;
double m_WasteMassCO = 0.0;
double m_WasteMassDust = 0.0;
double m_WasteMassBOD5 = 0.0;
double m_WasteMassCOD = 0.0;
double m_WasteMassTSS = 0.0;
double m_WasteMassTDS = 0.0;
double m_WasteMassSolids = 0.0;

// LCA Variables - EL of electricity
double m_ElectricityCoal = 0.0;
double m_ElectricityNaturalGas = 0.0;
double m_ElectricityOil = 0.0;
double m_ElectricityCO2 = 0.0;
double m_ElectricityNOx = 0.0;
double m_ElectricityVOC = 0.0;
double m_ElectricityCO = 0.0;
double m_ElectricityDust = 0.0;
double m_ElectricityBOD5 = 0.0;
double m_ElectricityCOD = 0.0;
double m_ElectricityTSS = 0.0;
double m_ElectricityTDS = 0.0;
double m_ElectricitySolids = 0.0;

// LCA Variables - EL of steam (heating duty)
double m_SteamCoal = 0.0;
double m_SteamNaturalGas = 0.0;
double m_SteamOil = 0.0;
double m_SteamCO2 = 0.0;
double m_SteamNOx = 0.0;
double m_SteamVOC = 0.0;
double m_SteamCO = 0.0;
double m_SteamDust = 0.0;
double m_SteamBOD5 = 0.0;
double m_SteamCOD = 0.0;
double m_SteamTSS = 0.0;
double m_SteamTDS = 0.0;
double m_SteamSolids = 0.0;

// FIND FEED AND PRODUCT STREAMS
shead = stream_head;
Find_Ass_Stream( unitPtr, assFeedStr, shead, FEED, &FeedStrNum );
Find_Ass_Stream( unitPtr, assProdStr, shead, PROD, &ProdStrNum );

// TOTAL ENVIRONMENTAL LOAD OF THE FEED STREAMS
for(i=0; i<FeedStrNum; i++)
{
    m_FeedMassCoal += assFeedStr[i]->m_MassCoal*assFeedStr[i]->MassFlow;
    m_FeedMassNaturalGas += assFeedStr[i]->m_MassNaturalGas*
                          assFeedStr[i]->MassFlow;
    m_FeedMassOil += assFeedStr[i]->m_MassOil*assFeedStr[i]->MassFlow;
    m_FeedMassCO2 += assFeedStr[i]->m_MassCO2*assFeedStr[i]->MassFlow;
    m_FeedMassNOx += assFeedStr[i]->m_MassNOx*assFeedStr[i]->MassFlow;
    m_FeedMassVOC += assFeedStr[i]->m_MassVOC*assFeedStr[i]->MassFlow;
    m_FeedMassCO += assFeedStr[i]->m_MassCO*assFeedStr[i]->MassFlow;
    m_FeedMassDust += assFeedStr[i]->m_MassDust*assFeedStr[i]->MassFlow;
    m_FeedMassBOD5 += assFeedStr[i]->m_MassBOD5*assFeedStr[i]->MassFlow;
    m_FeedMassCOD += assFeedStr[i]->m_MassCOD*assFeedStr[i]->MassFlow;
    m_FeedMassTSS += assFeedStr[i]->m_MassTSS*assFeedStr[i]->MassFlow;
    m_FeedMassTDS += assFeedStr[i]->m_MassTDS*assFeedStr[i]->MassFlow;
    m_FeedMassSolids += assFeedStr[i]->m_MassSolids*assFeedStr[i]->MassFlow;
}

shead = shead->next;

// TOTAL ENVIRONMENTAL LOAD OF THE WASTE STREAMS
for(i=0; i<ProdStrNum; i++)
{
    // CHECK IF TARGET STREAM IS AN INTERMEDIATE STREAM
    CheckForIntermediateStream(shead);

    if(!shead->m_LCAProduct)
    {
        m_WasteMassCoal += assProdStr[i]->m_MassCoal*
                           assProdStr[i]->MassFlow;
        m_WasteMassNaturalGas += assProdStr[i]->m_MassNaturalGas*
                                assProdStr[i]->MassFlow;
        m_WasteMassOil += assProdStr[i]->m_MassOil*assProdStr[i]->MassFlow;
        m_WasteMassCO2 += assProdStr[i]->m_MassCO2*assProdStr[i]->MassFlow;
        m_WasteMassNOx += assProdStr[i]->m_MassNOx*assProdStr[i]->MassFlow;
        m_WasteMassVOC += assProdStr[i]->m_MassVOC*assProdStr[i]->MassFlow;
        m_WasteMassCO += assProdStr[i]->m_MassCO*assProdStr[i]->MassFlow;
        m_WasteMassDust += assProdStr[i]->m_MassDust*assProdStr[i]->MassFlow;
        m_WasteMassBOD5 += assProdStr[i]->m_MassBOD5*assProdStr[i]->MassFlow;
        m_WasteMassCOD += assProdStr[i]->m_MassCOD*assProdStr[i]->MassFlow;
        m_WasteMassTSS += assProdStr[i]->m_MassTSS*assProdStr[i]->MassFlow;
        m_WasteMassTDS += assProdStr[i]->m_MassTDS*assProdStr[i]->MassFlow;
        m_WasteMassSolids += assProdStr[i]->m_MassSolids*assProdStr[i]->MassFlow;
    }

    shead = shead->next;
}

// TOTAL ENVIRONMENTAL LOAD OF ELECTRICITY INPUT (POWER + COOLING DUTY)
m_ElectricityCoal = (unitPtr->Power + unitPtr->Duty[1]*4.184)*
                    LCAPtr->m_EnergyCoal[1];
m_ElectricityNaturalGas = (unitPtr->Power + unitPtr->Duty[1]*4.184)*
                          LCAPtr->m_EnergyNaturalGas[1];
m_ElectricityOil = (unitPtr->Power + unitPtr->Duty[1]*4.184)*
LCAPtr->m_EnergyOil[1];

m_ElectricityCO2 = (unitPtr->Power + unitPtr->Duty[1]*4.184)*
LCAPtr->m_EnergyCO2[1];

m_ElectricityNOx = (unitPtr->Power + unitPtr->Duty[1]*4.184)*
LCAPtr->m_EnergyNOx[1];

m_ElectricityVOC = (unitPtr->Power + unitPtr->Duty[1]*4.184)*
LCAPtr->m_EnergyVOC[1];

m_ElectricityCO = (unitPtr->Power + unitPtr->Duty[1]*4.184)*
LCAPtr->m_EnergyCO[1];

m_ElectricityDust = (unitPtr->Power + unitPtr->Duty[1]*4.184)*
LCAPtr->m_EnergyDust[1];

m_ElectricityBOD5 = (unitPtr->Power + unitPtr->Duty[1]*4.184)*
LCAPtr->m_EnergyBOD5[1];

m_ElectricityCOD = (unitPtr->Power + unitPtr->Duty[1]*4.184)*
LCAPtr->m_EnergyCOD[1];

m_ElectricityTSS = (unitPtr->Power + unitPtr->Duty[1]*4.184)*
LCAPtr->m_EnergyTSS[1];

m_ElectricityTDS = (unitPtr->Power + unitPtr->Duty[1]*4.184)*
LCAPtr->m_EnergyTDS[1];

m_ElectricitySolids = (unitPtr->Power + unitPtr->Duty[1]*4.184)*
LCAPtr->m_EnergySolids[1];

// TOTAL ENVIRONMENTAL LOAD OF STEAM INPUT (HEATING DUTY)

m_SteamCoal = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyCoal[0];

m_SteamNaturalGas = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyNaturalGas[0];

m_SteamOil = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyOil[0];

m_SteamCO2 = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyCO2[0];

m_SteamNOx = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyNOx[0];

m_SteamVOC = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyVOC[0];

m_SteamCO = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyCO[0];

m_SteamDust = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyDust[0];

m_SteamBOD5 = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyBOD5[0];

m_SteamCOD = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyCOD[0];

m_SteamTSS = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyTSS[0];

m_SteamTDS = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyTDS[0];

m_SteamSolids = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergySolids[0];

// FIND ALL UNIT_OP PRODUCT STREAMS

shead = stream_head;
Find_Ass_Stream( unitPtr, assProdStr, shead, PROD, &ProdStrNum );

// ASSIGN TOTAL ENVIRONMENTAL LOAD TO THE PRODUCT STREAMS

// MASS BASED SPLIT

// FIND TOTAL MASS OF PRODUCT STREAMS

m_totalProductMass = 0.0;
for(i=0; i<ProdStrNum; i++)
{
    if(assProdStr[i]->m_LCAProduct)
        m_totalProductMass += assProdStr[i]->MassFlow;
}

// FIND SPLIT MASS FRACTIONS FOR EACH PRODUCT STREAM

for(i=0; i<ProdStrNum; i++)
{
    if(assProdStr[i]->m_LCAProduct)
if(assProdStr[i]->m_LCAProduct)
    m_splitMassFraction[i] = assProdStr[i]->MassFlow/
m_totalProductMass;
else
    m_splitMassFraction[i] = 0.0;
}

for(i=0; i<ProdStrNum; i++)
{
    if(assProdStr[i]->m_LCAProduct)
    {
        // The data for the env. load of the waste streams have to be
        // negative because in the following expressions we use the
        // notation (m_FeedImpact - m_WasteImpact)

        if(assProdStr[i]->MassFlow > 0.0) // Avoid division by 0
        {
            assProdStr[i]->m_MassCoal = m_splitMassFraction[i]*
            (m_FeedMassCoal - m_WasteMassCoal + m_ElectricityCoal +
             m_SteamCoal)/assProdStr[i]->MassFlow;
            assProdStr[i]->m_MassNaturalGas = m_splitMassFraction[i]*
            (m_FeedMassNaturalGas - m_WasteMassNaturalGas +
             m_ElectricityNaturalGas + m_SteamNaturalGas)/
            assProdStr[i]->MassFlow;
            assProdStr[i]->m_MassOil = m_splitMassFraction[i]*
            (m_FeedMassOil - m_WasteMassOil + m_ElectricityOil +
             m_SteamOil)/assProdStr[i]->MassFlow;
            assProdStr[i]->m_MassCO2 = m_splitMassFraction[i]*
            (m_FeedMassCO2 - m_WasteMassCO2 + m_ElectricityCO2 +
             m_SteamCO2)/assProdStr[i]->MassFlow;
            assProdStr[i]->m_MassNOx = m_splitMassFraction[i]*
            (m_FeedMassNOx - m_WasteMassNOx + m_ElectricityNOx +
             m_SteamNOx)/assProdStr[i]->MassFlow;
            assProdStr[i]->m_MassVOC = m_splitMassFraction[i]*
            (m_FeedMassVOC - m_WasteMassVOC + m_ElectricityVOC +
             m_SteamVOC)/assProdStr[i]->MassFlow;
            assProdStr[i]->m_MassCO = m_splitMassFraction[i]*
            (m_FeedMassCO - m_WasteMassCO + m_ElectricityCO +
             m_SteamCO)/assProdStr[i]->MassFlow;
            assProdStr[i]->m_MassDust = m_splitMassFraction[i]*
            (m_FeedMassDust - m_WasteMassDust + m_ElectricityDust +
             m_SteamDust)/assProdStr[i]->MassFlow;
            assProdStr[i]->m_MassBOD5 = m_splitMassFraction[i]*
            (m_FeedMassBOD5 - m_WasteMassBOD5 + m_ElectricityBOD5 +
             m_SteamBOD5)/assProdStr[i]->MassFlow;
            assProdStr[i]->m_MassCOD = m_splitMassFraction[i]*
            (m_FeedMassCOD - m_WasteMassCOD + m_ElectricityCOD +
             m_SteamCOD)/assProdStr[i]->MassFlow;
            assProdStr[i]->m_MassTSS = m_splitMassFraction[i]*
            (m_FeedMassTSS - m_WasteMassTSS + m_ElectricityTSS +
             m_SteamTSS)/assProdStr[i]->MassFlow;
            assProdStr[i]->m_MassTDS = m_splitMassFraction[i]*
            (m_FeedMassTDS - m_WasteMassTDS + m_ElectricityTDS +
             m_SteamTDS)/assProdStr[i]->MassFlow;
            assProdStr[i]->m_MassSolids = m_splitMassFraction[i]*
        }
    }
}
(m_FeedMassSolids - m_WasteMassSolids + m_ElectricitySolids
m_SteamSolids)/assProdStr[i]->MassFlow;
)
)
)

/***************************************************************************/
* doLCAReport: generates the LCA report and saves the                 *
* results in a file                                                  *
/***************************************************************************/

int doLCAReport(void)
{
  FILE *fp;
  SPReply theReply;
  Str255 fileName, windName;
  char cString[50];
  Str255 pString;
  short theCount, i;
  char MyFile[64];
  OSErr theError = noErr;

  if (stream_head == NULL)
    return;

  GetWTitle(flowWindow, windName);
  myPtoCstr(windName, MyFile);
  theCount = strlen(MyFile);

  i = 0;
  while (MyFile[i] != '.' && i < theCount) {
    i++;
  }

  MyFile[i] = '.';
  MyFile[i+1] = 'E';
  MyFile[i+2] = 'L';
  MyFile[i+3] = '\0';

  myCtoPstr(MyFile, fileName);

  strcpy(cString, "Save Inventory Report As...");
  myCtoPstr(cString, pString);

  if(!CreateSaveFile(&theReply, fileName, pString))
    return;

  SetCursor(&delayCursor);

  myPtoCstr(theReply.fName, MyFile);

  if((fp = fopen(MyFile, "w")) == NULL) {
    /* postErr(NULL); */
    return;
  }
LCAReport( FILE *fp )
{
    LINK unitPtr = flow_head;
    St_LINK shead = stream_head;
    St_LINK stream[6];
    St_LINK assFeedStr[NUM_OF_MAX_FEED_STREAMS+2];
    St_LINK assProdStr[NUM_OF_MAX_FEED_STREAMS+2];
    short i, j, FeedStrNum, ProdStrNum;
    short theNumber;
    short RAW_NUM;  // # of streams printed in parallel
    short UPPER_LIMIT;
    LINK toUnitPtr, fromUnitPtr;

    theNumber = NumOfStreams;
    RAW_NUM = 5;

    fprintf( fp, "LIFE CYCLE INVENTORY ANALYSIS REPORT\n\n\n\n" );

    while( theNumber > 0 )
    {
        if( theNumber < 5 )
            RAW_NUM = theNumber;

        for(i=0; i<RAW_NUM; i++)
        {
            stream[i] = shead;
            shead = shead->next;
        }

        RAW_NUM = i;
        UPPER_LIMIT = RAW_NUM;
        if( UPPER_LIMIT < 2 ) UPPER_LIMIT = 2;

        fprintf( fp, "------------------------" );
        for(j=0; j<UPPER_LIMIT; j++)
            fprintf( fp, "-" );
        fprintf( fp, "\n" );

        fprintf( fp, "MASS ECO-VECTOR\t\t" );
        for(j=0; j<RAW_NUM; j++)
            fprintf( fp, "%s\t\t", stream[j]->name );
        fprintf( fp, "\n\n" );
fprintf( fp, "%8.2e	", stream[j]->m_MassCoal );
fprintf( fp, "\n" );

fprintf( fp, "%8.2e	", stream[j]->m_MassNaturalGas );
fprintf( fp, "\n" );

fprintf( fp, "%8.2e	", stream[j]->m_MassOil );
fprintf( fp, "\n" );

fprintf( fp, "%8.2e	", stream[j]->m_MassCO2 );
fprintf( fp, "\n" );

fprintf( fp, "%8.2e	", stream[j]->m_MassNOx );
fprintf( fp, "\n" );

fprintf( fp, "%8.2e	", stream[j]->m_MassVOC );
fprintf( fp, "\n" );

fprintf( fp, "%8.2e	", stream[j]->m_MassCO );
fprintf( fp, "\n" );

fprintf( fp, "%8.2e	", stream[j]->m_MassDust );
fprintf( fp, "\n" );

fprintf( fp, "%8.2e	", stream[j]->m_MassBOD5 );
fprintf( fp, "\n" );

fprintf( fp, "%8.2e	", stream[j]->m_MassCOD );
fprintf( fp, "\n" );

fprintf( fp, "%8.2e	", stream[j]->m_MassTSS );
fprintf( fp, "\n" );

fprintf( fp, "%8.2e	", stream[j]->m_MassTDS );
fprintf( fp, "\n" );
<table>
<thead>
<tr>
<th>Solid Waste</th>
<th>Mass</th>
<th>Steam</th>
<th>ECO-VECTOR</th>
<th>STEAM</th>
<th>ELECTRICITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Natural Gas</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oil</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CO₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOₓ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
fprintf( fp, "\n" );

fprintf( fp, "	Dust	\t\t\t" );
for(j=0; j<2; j++)
    fprintf( fp, "%.8e\t", LCAPtr->m_EnergyDust[j] );
fprintf( fp, "\n" );

fprintf( fp, "	BOD5	\t\t\t" );
for(j=0; j<2; j++)
    fprintf( fp, "%.8e\t", LCAPtr->m_EnergyBOD5[j] );
fprintf( fp, "\n" );

fprintf( fp, "	COD	\t\t\t" );
for(j=0; j<2; j++)
    fprintf( fp, "%.8e\t", LCAPtr->m_EnergyCOD[j] );
fprintf( fp, "\n" );

fprintf( fp, "	TSS	\t\t\t" );
for(j=0; j<2; j++)
    fprintf( fp, "%.8e\t", LCAPtr->m_EnergyTSS[j] );
fprintf( fp, "\n" );

fprintf( fp, "	TDS	\t\t\t" );
for(j=0; j<2; j++)
    fprintf( fp, "%.8e\t", LCAPtr->m_EnergyTDS[j] );
fprintf( fp, "\n" );

fprintf( fp, "	Solid Waste	" );
for(j=0; j<2; j++)
    fprintf( fp, "%.8e\t", LCAPtr->m_EnergySolids[j] );
fprintf( fp, "\n\n\n" );

// REPORT ENVIRONMENTAL LOADS FOR EACH UNIT OP

while( unitPtr != NULL)
{
    UnitOpReport( fp, unitPtr);
    unitPtr = unitPtr->next;
}

UnitOpReport(FILE *fp, LINK unitPtr)
{
    St_LINK shead;
    St_LINK assFeedStr[NUM_OF_MAX_FEED_STREAMS+2];
    St_LINK assProdStr[NUM_OF_MAX_FEED_STREAMS+2];
    short i, j, FeedStrNum, ProdStrNum;

double m_FeedMassCoal = 0.0; // LCA Variables - EL of Feed Stream
double m_FeedMassNaturalGas = 0.0;
double m_FeedMassOil = 0.0;
double m_FeedMassCO2 = 0.0;
double m_FeedMassNOx = 0.0;
double m_FeedMassVOC = 0.0;
double m_FeedMassCO = 0.0;
double m_FeedMassDust = 0.0;
double m_FeedMassBOD5 = 0.0;
double m_FeedMassCOD = 0.0;
double m_FeedMassTSS = 0.0;
double m_FeedMassTDS = 0.0;
double m_FeedMassSolids = 0.0;

double m_WasteMassCoal = 0.0; // LCA Variables - EL of Waste Streams
double m_WasteMassNaturalGas = 0.0;
double m_WasteMassOil = 0.0;
double m_WasteMassCO2 = 0.0;
double m_WasteMassNOx = 0.0;
double m_WasteMassVOC = 0.0;
double m_WasteMassCO = 0.0;
double m_WasteMassDust = 0.0;
double m_WasteMassBOD5 = 0.0;
double m_WasteMassCOD = 0.0;
double m_WasteMassTSS = 0.0;
double m_WasteMassTDS = 0.0;
double m_WasteMassSolids = 0.0;

double m_ElectricityCoal = 0.0; // LCA Variables - EL of electricity input 
// (power + cooling duty)
double m_ElectricityNaturalGas = 0.0;
double m_ElectricityOil = 0.0;
double m_ElectricityCO2 = 0.0;
double m_ElectricityNOx = 0.0;
double m_ElectricityVOC = 0.0;
double m_ElectricityCO = 0.0;
double m_ElectricityDust = 0.0;
double m_ElectricityBOD5 = 0.0;
double m_ElectricityCOD = 0.0;
double m_ElectricityTSS = 0.0;
double m_ElectricityTDS = 0.0;
double m_ElectricitySolids = 0.0;

double m_SteamCoal = 0.0; // LCA Variables - EL of steam (heating duty)
double m_SteamNaturalGas = 0.0;
double m_SteamOil = 0.0;
double m_SteamCO2 = 0.0;
double m_SteamNOx = 0.0;
double m_SteamVOC = 0.0;
double m_SteamCO = 0.0;
double m_SteamDust = 0.0;
double m_SteamBOD5 = 0.0;
double m_SteamCOD = 0.0;
double m_SteamTSS = 0.0;
double m_SteamTDS = 0.0;
double m_SteamSolids = 0.0;

// FIND FEED AND PRODUCT STREAMS
shead = stream_head;
Find_Ass_Stream( unitPtr, assFeedStr, shead, FEED, &FeedStrNum );
Find_Ass_Stream( unitPtr, assProdStr, shead, PROD, &ProdStrNum );
for (i = 0; i < FeedStrNum; i++)
{
    m_FeedMassCoal += assFeedStr[i]->m_MassCoal * 
                    assFeedStr[i]->MassFlow;

    m_FeedMassNaturalGas += assFeedStr[i]->m_MassNaturalGas * 
                           assFeedStr[i]->MassFlow;

    m_FeedMassOil += assFeedStr[i]->m_MassOil * 
                    assFeedStr[i]->MassFlow;

    m_FeedMassCO2 += assFeedStr[i]->m_MassCO2 * 
                    assFeedStr[i]->MassFlow;

    m_FeedMassNOx += assFeedStr[i]->m_MassNOx * 
                    assFeedStr[i]->MassFlow;

    m_FeedMassVOC += assFeedStr[i]->m_MassVOC * 
                    assFeedStr[i]->MassFlow;

    m_FeedMassCO += assFeedStr[i]->m_MassCO * 
                    assFeedStr[i]->MassFlow;

    m_FeedMassDust += assFeedStr[i]->m_MassDust * 
                     assFeedStr[i]->MassFlow;

    m_FeedMassBOD5 += assFeedStr[i]->m_MassBOD5 * 
                     assFeedStr[i]->MassFlow;

    m_FeedMassCOD += assFeedStr[i]->m_MassCOD * 
                    assFeedStr[i]->MassFlow;

    m_FeedMassTSS += assFeedStr[i]->m_MassTSS * 
                    assFeedStr[i]->MassFlow;

    m_FeedMassTDS += assFeedStr[i]->m_MassTDS * 
                    assFeedStr[i]->MassFlow;

    m_FeedMassSolids += assFeedStr[i]->m_MassSolids * 
                       assFeedStr[i]->MassFlow;

    shead = shead->next;
}

for (i = 0; i < ProdStrNum; i++)
{
    // Check if target stream is an intermediate stream
    CheckForIntermediateStream(shead);

    if (!shead->m_LCAProduct)
    {
        m_WasteMassCoal += assProdStr[i]->m_MassCoal * 
                            assProdStr[i]->MassFlow;

        m_WasteMassNaturalGas += assProdStr[i]->m_MassNaturalGas * 
                                assProdStr[i]->MassFlow;

        m_WasteMassOil += assProdStr[i]->m_MassOil * 
                         assProdStr[i]->MassFlow;

        m_WasteMassCO2 += assProdStr[i]->m_MassCO2 * 
                         assProdStr[i]->MassFlow;

        m_WasteMassNOx += assProdStr[i]->m_MassNOx * 
                         assProdStr[i]->MassFlow;

        m_WasteMassVOC += assProdStr[i]->m_MassVOC * 
                         assProdStr[i]->MassFlow;

        m_WasteMassCO += assProdStr[i]->m_MassCO * 
                         assProdStr[i]->MassFlow;

        m_WasteMassDust += assProdStr[i]->m_MassDust * 
                           assProdStr[i]->MassFlow;

        m_WasteMassBOD5 += assProdStr[i]->m_MassBOD5 * 
                          assProdStr[i]->MassFlow;

        m_WasteMassCOD += assProdStr[i]->m_MassCOD * 
                         assProdStr[i]->MassFlow;

        m_WasteMassTSS += assProdStr[i]->m_MassTSS * 
                         assProdStr[i]->MassFlow;

        m_WasteMassTDS += assProdStr[i]->m_MassTDS * 
                         assProdStr[i]->MassFlow;
    }
}
assProdStr[i]->MassFlow;

m_WasteMassSolids += assProdStr[i]->m_MassSolids*assProdStr[i]->MassFlow;
}
shead = shead->next;

// TOTAL ENVIRONMENTAL LOAD OF ELECTRICITY INPUT (POWER + COOLING DUTY)
m_ElectricityCoal = (unitPtr->Power + unitPtr->Duty[1]*4.184)*LCAPtr->m_EnergyCoal[1];
m_ElectricityNaturalGas = (unitPtr->Power + unitPtr->Duty[1]*4.184)*LCAPtr->m_EnergyNaturalGas[1];
m_ElectricityOil = (unitPtr->Power + unitPtr->Duty[1]*4.184)*LCAPtr->m_EnergyOil[1];
m_ElectricityCO2 = (unitPtr->Power + unitPtr->Duty[1]*4.184)*LCAPtr->m_EnergyCO2[1];
m_ElectricityNOx = (unitPtr->Power + unitPtr->Duty[1]*4.184)*LCAPtr->m_EnergyNOx[1];
m_ElectricityVOC = (unitPtr->Power + unitPtr->Duty[1]*4.184)*LCAPtr->m_EnergyVOC[1];
m_ElectricityCO = (unitPtr->Power + unitPtr->Duty[1]*4.184)*LCAPtr->m_EnergyCO[1];
m_ElectricityDust = (unitPtr->Power + unitPtr->Duty[1]*4.184)*LCAPtr->m_EnergyDust[1];
m_ElectricityBOD5 = (unitPtr->Power + unitPtr->Duty[1]*4.184)*LCAPtr->m_EnergyBOD5[1];
m_ElectricityCOD = (unitPtr->Power + unitPtr->Duty[1]*4.184)*LCAPtr->m_EnergyCOD[1];
m_ElectricityTSS = (unitPtr->Power + unitPtr->Duty[1]*4.184)*LCAPtr->m_EnergyTSS[1];
m_ElectricityTDS = (unitPtr->Power + unitPtr->Duty[1]*4.184)*LCAPtr->m_EnergyTDS[1];
m_ElectricitySolids = (unitPtr->Power + unitPtr->Duty[1]*4.184)*LCAPtr->m_EnergySolids[1];

// TOTAL ENVIRONMENTAL LOAD OF STEAM INPUT (HEATING DUTY)
m_SteamCoal = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyCoal[0];
m_SteamNaturalGas = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyNaturalGas[0];
m_SteamOil = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyOil[0];
m_SteamCO2 = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyCO2[0];
m_SteamNOx = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyNOx[0];
m_SteamVOC = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyVOC[0];
m_SteamCO = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyCO[0];
m_SteamDust = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyDust[0];
m_SteamBOD5 = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyBOD5[0];
m_SteamCOD = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyCOD[0];
m_SteamTSS = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyTSS[0];
m_SteamTDS = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergyTDS[0];
m_SteamSolids = (unitPtr->Duty[0]*4.184)*LCAPtr->m_EnergySolids[0];

fprintf(fp, "%s\tENVIRONMENTAL LOADS (kg/h) FROM\n", unitPtr->name);
fprintf(fp, "\t\tFEEDS\t\tWASTES\t\tSTEAM\t\tELECTRICITY\n");

fprintf(fp, \"\t\t\t%8.2e\t%8.2e\t%8.2e\t%8.2e\n\", "Coal", m_FeedMassCoal, m_WasteMassCoal, m_SteamCoal, m_ElectricityCoal);
fprintf( fp, "\t\t\t%8.2e\t%8.2e\t%8.2e\t%8.2e\n", "NaturalGas", m_FeedMassNaturalGas, m_WasteMassNaturalGas, m_SteamNaturalGas, m_ElectricityNaturalGas);

fprintf( fp, "\t\t\t%8.2e\t%8.2e\t%8.2e\t%8.2e\n", "Oil", m_FeedMassOil, m_WasteMassOil, m_SteamOil, m_ElectricityOil);

fprintf( fp, "\t\t\t%8.2e\t%8.2e\t%8.2e\t%8.2e\n", "CO2", m_FeedMassCO2, m_WasteMassCO2, m_SteamCO2, m_ElectricityCO2);

fprintf( fp, "\t\t\t%8.2e\t%8.2e\t%8.2e\t%8.2e\n", "NOx", m_FeedMassNOx, m_WasteMassNOx, m_SteamNOx, m_ElectricityNOx);

fprintf( fp, "\t\t\t%8.2e\t%8.2e\t%8.2e\t%8.2e\n", "VOC", m_FeedMassVOC, m_WasteMassVOC, m_SteamVOC, m_ElectricityVOC);

fprintf( fp, "\t\t\t%8.2e\t%8.2e\t%8.2e\t%8.2e\n", "CO", m_FeedMassCO, m_WasteMassCO, m_SteamCO, m_ElectricityCO);

fprintf( fp, "\t\t\t%8.2e\t%8.2e\t%8.2e\t%8.2e\n", "Dust", m_FeedMassDust, m_WasteMassDust, m_SteamDust, m_ElectricityDust);

fprintf( fp, "\t\t\t%8.2e\t%8.2e\t%8.2e\t%8.2e\n", "BOD5", m_FeedMassBOD5, m_WasteMassBOD5, m_SteamBOD5, m_ElectricityBOD5);

fprintf( fp, "\t\t\t%8.2e\t%8.2e\t%8.2e\t%8.2e\n", "COD", m_FeedMassCOD, m_WasteMassCOD, m_SteamCOD, m_ElectricityCOD);

fprintf( fp, "\t\t\t%8.2e\t%8.2e\t%8.2e\t%8.2e\n", "TSS", m_FeedMassTSS, m_WasteMassTSS, m_SteamTSS, m_ElectricityTSS);

fprintf( fp, "\t\t\t%8.2e\t%8.2e\t%8.2e\t%8.2e\n", "TDS", m_FeedMassTDS, m_WasteMassTDS, m_SteamTDS, m_ElectricityTDS);

fprintf( fp, "\t\t\t%8.2e\t%8.2e\t%8.2e\t%8.2e\n", "SolidWaste", m_FeedMassSolids, m_WasteMassSolids, m_SteamSolids, m_ElectricitySolids); fprintf( fp, "\n\n" );
APPENDIX V

ENVIRONMENTAL STREAM PROPERTIES IN ENVIROCAD
/* Function that estimated the environmental stream properties based on contribution of individual compounds * 
* K. Abeliotis May 1993 * 
*/

void Estimate_EnviroStreamProperties(St_LINK streamPtr)
{
    St_CompPtr SCompPtr = streamPtr->StCompList;
    double Q, massFlow;
    short i;

    Q = Estim_VolFlow(streamPtr); // m3/h
    streamPtr->TOC_Stream = 0.0; // All stream properties are estimated in mg/L
    streamPtr->COD_Stream = 0.0;
    streamPtr->ThOD_Stream = 0.0;
    streamPtr->BODu_Stream = 0.0;
    streamPtr->BOD5_Stream = 0.0;
    streamPtr->TKN_Stream = 0.0;
    streamPtr->NH3_Stream = 0.0;
    streamPtr->NO3_NO2_Stream = 0.0;
    streamPtr->TP_Stream = 0.0;
    streamPtr->TS_Stream = 0.0;
    streamPtr->TSS_Stream = 0.0;
    streamPtr->VSS_Stream = 0.0;
    streamPtr->TS_Stream = 0.0;
    streamPtr->TDS_Stream = 0.0;
    streamPtr->VDS_Stream = 0.0;
    streamPtr->DVDS_Stream = 0.0;

    for(i=0; i<NumOfComps; i++)
    {
        massFlow = SCompPtr->CompMassFlow; // kg/h
        if(massFlow > 1.0e-8)
        {
            streamPtr->TOC_Stream += (massFlow*theComps[i]->TOC*1000.0)/Q;
            streamPtr->COD_Stream += (massFlow*theComps[i]->COD*1000.0)/Q;
            streamPtr->ThOD_Stream += (massFlow*theComps[i]->ThOD*1000.0)/Q;
            streamPtr->BODu_Stream += (massFlow*theComps[i]->BOD_COD_Ratio*1000.0)/Q;
            streamPtr->BOD5_Stream += (massFlow*theComps[i]->BOD_COD_Ratio*1000.0)/Q;
            streamPtr->TKN_Stream += (massFlow*theComps[i]->TKN*1000.0)/Q;
            streamPtr->NH3_Stream += (massFlow*theComps[i]->NH3_Fraction*1000.0)/Q;
            streamPtr->NO3_NO2_Stream += (massFlow*theComps[i]->NO3_NO2*1000.0)/Q;
        }
    }
streamPtr->TP_Stream += (massFlow*theComps[i]->TP*1000.0)/Q;

streamPtr->TS_Stream += (massFlow*theComps[i]->TS*1000.0)/Q;
streamPtr->TSS_Stream += (massFlow*theComps[i]->TS*theComps[i]->TSS_TS_Ratio*1000.0)/Q;
streamPtr->VSS_Stream += (massFlow*theComps[i]->TS*theComps[i]->TSS_TS_Ratio*theComps[i]->VSS_TSS_Ratio*1000.0)/Q;
streamPtr->DVSS_Stream += (massFlow*theComps[i]->TS*theComps[i]->TSS_TS_Ratio*theComps[i]->VSS_TSS_Ratio*theComps[i]->DVSS_VSS_Ratio*1000.0)/Q;
streamPtr->TDS_Stream += (massFlow*theComps[i]->TS*(1.0 - theComps[i]->TSS_TS_Ratio)*1000.0)/Q;
streamPtr->VDS_Stream += (massFlow*theComps[i]->TS*(1.0 - theComps[i]->TSS_TS_Ratio)*theComps[i]->VDS_TDS_Ratio*1000.0)/Q;
streamPtr->DVDS_Stream += (massFlow*theComps[i]->TS*(1.0 - theComps[i]->TSS_TS_Ratio)*theComps[i]->VDS_TDS_Ratio*theComps[i]->DVDS_VDS_Ratio*1000.0)/Q;

SCompPtr = SCompPtr->next;
}
REFERENCES


