Reference Guide: ThermoeconomicModel and cThermoeconomicModel

TaesLab Version: 1.8
Generated: November 2025

Table of Contents

Overview

The **ThermoeconomicModel** system is the core interface of TaesLab for performing comprehensive thermoeconomic analysis. It consists of:

1. `ThermoeconomicModel()` - Base function that creates cThermoeconomicModel objects
2. `cThermoeconomicModel` - Main analysis class with all calculation methods
3. **Base functions** - Utility functions for displaying, saving, and exporting results

The system follows TaesLab's object-oriented architecture, where:

- **Data flows** from file → cDataModel → cThermoeconomicModel
- **Results** are generated as cResultInfo objects containing cTable collections
- **Presentation** is handled by Base functions that work with any cResultSet object

ThermoeconomicModel Function

Purpose: Create a cThermoeconomicModel object from a data model file or cDataModel object.

Syntax

```
model = ThermoeconomicModel(arg, Name, Value)
```

Input Arguments

Argument	Type	Description
`arg`	char/string/cDataModel	File path to data model or cDataModel object

Name-Value Arguments

Parameter	Type	Default	Description
`State`	char/string	First state	Operation state name
`ReferenceState`	char/string	First state	Reference state for diagnosis
`ResourceSample`	char/string	First sample	Resource sample name
`CostTables`	char	'ALL'	Cost tables output: 'DIRECT', 'GENERALIZED', 'ALL'
`DiagnosisMethod`	char	'NONE'	Diagnosis method: 'NONE', 'WASTE_EXTERNAL', 'WASTE_INTERNAL'
`ActiveWaste`	char/string	First waste	Active waste flow name
`Summary`	char	'NONE'	Summary results: 'NONE', 'STATES', 'RESOURCES', 'ALL'
`Recycling`	logical	false	Activate recycling analysis
`Debug`	logical	true	Print debug information

Output Arguments

Argument	Type	Description
`model`	cThermoeconomicModel	Configured thermoeconomic model object

Examples

Basic usage:

```
% From file
model = ThermoeconomicModel('rankine_model.json');

% From cDataModel object
data = ReadDataModel('plant_data.xlsx');
model = ThermoeconomicModel(data);
```

Advanced configuration:

```
model = ThermoeconomicModel('cogeneration.json', ...
    'State', 'design', ...
    'ResourceSample', 'summer_costs', ...
    'CostTables', 'ALL', ...
    'Summary', 'STATES', ...
    'Debug', false);
```

cThermoeconomicModel Class

The `cThermoeconomicModel` class is the main analysis engine that extends `cResultSet` and provides comprehensive thermoeconomic analysis capabilities.

Properties

Read-Only Properties

Property	Type	Description
`DataModel`	cDataModel	Source data model object
`StateNames`	cell array	Names of defined thermodynamic states
`SampleNames`	cell array	Names of defined resource cost samples
`WasteFlows`	cell array	Names of waste flows
`ResourceData`	cResourceData	Current resource cost data object

Configuration Properties

Property	Type	Description
`CurrentState`	char	Active operation state name
`CurrentSample`	char	Active resource sample name
`ReferenceState`	char	Reference state for diagnosis
`CostTables`	char	Selected cost result tables
`DiagnosisMethod`	char	Method for waste fuel impact calculation
`Summary`	char	Summary results selection
`Recycling`	logical	Recycling analysis status
`ActiveWaste`	char	Active waste flow for analysis

Configuration Methods

State Configuration

`setState(obj, stateName)`

```
% Set active operation state
model.setState('off_design');
```

setReferenceState(obj, stateName)

```
% Set reference state for diagnosis
model.setReferenceState('design');
```

Cost Configuration

setResourceSample(obj, sampleName)

```
% Set active resource cost sample
model.setResourceSample('winter_costs');
```

setCostTables(obj, costTables)

```
% Configure cost table generation
model.setCostTables('ALL');  % 'DIRECT', 'GENERALIZED', or 'ALL'
```

Analysis Configuration

setDiagnosisMethod(obj, method)

```
% Set diagnosis method
model.setDiagnosisMethod('WASTE_INTERNAL');
```

setActiveWaste(obj, wasteName)

```
% Set active waste for analysis
model.setActiveWaste('QC');
```

setRecycling(obj, status)

```
% Enable/disable recycling analysis
model.setRecycling(true);
```

setSummary(obj, summaryType)

```
% Configure summary results
model.setSummary('STATES');  % 'NONE', 'STATES', 'RESOURCES', 'ALL'
```

Debug Configuration

setDebug(obj, status)

```
% Enable/disable debug mode
model.setDebug(false);
```

`toggleDebug(obj)`

```
% Toggle debug mode
model.toggleDebug();
```

Analysis Methods

All analysis methods return `cResultInfo` objects containing organized result tables.

Core Analysis

`productiveStructure(obj)` **Purpose**: Get productive structure analysis
Returns: cResultInfo with flow-process relationships

```
structure = model.productiveStructure();
ShowResults(structure);
```

`exergyAnalysis(obj)` **Purpose**: Perform exergy analysis
Returns: cResultInfo with exergy flows, destructions, and efficiencies

```
exergyResults = model.exergyAnalysis();
ShowResults(exergyResults, 'Table', 'efficiency');
```

`thermoeconomicAnalysis(obj)` **Purpose**: Perform complete thermoeconomic analysis
Returns: cResultInfo with cost tables, cost formation, and economic indicators

```
thermoResults = model.thermoeconomicAnalysis();
ShowResults(thermoResults, 'Table', 'dcost');
```

Specialized Analysis

`wasteAnalysis(obj)` **Purpose**: Analyze waste cost allocation and recycling
Returns: cResultInfo with waste cost analysis

```
if model.isWaste()
    wasteResults = model.wasteAnalysis();
    ShowResults(wasteResults);
end
```

`thermoeconomicDiagnosis(obj)` **Purpose**: Compare states for malfunction detection
Returns: cResultInfo with diagnosis indicators

```
if model.isDiagnosis()
    diagnosis = model.thermoeconomicDiagnosis();
    ShowResults(diagnosis);
end
```

Diagram Generation

`productiveDiagram(obj)` **Purpose**: Generate productive structure diagrams
Returns: cResultInfo with diagram data

```
diagrams = model.productiveDiagram();
ShowResults(diagrams, 'Table', 'adjacency');
```

`diagramFP(obj)` **Purpose**: Generate Fuel-Product diagrams
Returns: cResultInfo with F-P diagram matrices

```
fpDiagrams = model.diagramFP();
ShowResults(fpDiagrams);
```

Summary Analysis

`summaryResults(obj)` **Purpose**: Generate summary comparison tables
Returns: cResultInfo with comparative analysis

```
if model.isSummaryActive()
    summary = model.summaryResults();
    ShowResults(summary);
end
```

`dataInfo(obj)` **Purpose**: Get data model information
Returns: cResultInfo with data model tables

```
dataInfo = model.dataInfo();
ShowResults(dataInfo);
```

Information Methods

Generic Result Access

`getResultInfo(obj, type)` **Purpose**: Get specific result type
Parameters:

- `type` - Result type identifier

`showResultInfo(obj)` **Purpose**: Display all available result types
Returns: Structure with result type information

```
availableResults = model.showResultInfo();
disp(availableResults);
```

Model Information

`showProperties(obj)` **Purpose**: Display current model configuration
Returns: Structure with model properties

```
properties = model.showProperties();
```

Table Information

`getTablesDirectory(obj)` **Purpose**: Get directory of available tables
Returns: cTable with table directory

```
directory = model.getTablesDirectory();
ShowTable(directory);
```

`getTableInfo(obj, tableName)`

Purpose: Get information about specific table
Returns: Structure with table metadata

```
info = model.getTableInfo('dcost');
```

Validation Methods

Resource Cost Validation

`isResourceCost(obj)` **Purpose**: Check if model has resource cost data
Returns: logical

```
if model.isResourceCost()
    % Perform cost analysis
end
```

Cost Table Validation

`isDirectCost(obj)` **Purpose**: Check if direct cost tables are enabled
Returns: logical

`isGeneralCost(obj)` **Purpose**: Check if generalized cost tables are enabled
Returns: logical

Analysis Validation

`isDiagnosis(obj)` **Purpose**: Check if diagnosis analysis is possible
Returns: logical

```
if model.isDiagnosis()
    diagnosis = model.thermoeconomicDiagnosis();
end
```

`isWaste(obj)` **Purpose**: Check if model has waste flows
Returns: logical

Summary Validation

`summaryOptions(obj)` **Purpose**: Get available summary options
Returns: Structure with summary capabilities

`isSummaryEnable(obj)` **Purpose**: Check if summary results are available
Returns: logical

`isSummaryActive(obj)` **Purpose**: Check if summary is currently activated
Returns: logical

`isStateSummary(obj)` **Purpose**: Check if state summary is available
Returns: logical

`isSampleSummary(obj)` **Purpose**: Check if sample summary is available
Returns: logical

Inherited cResultSet Methods

As a subclass of `cResultSet`, `cThermoeconomicModel` inherits all result management methods:

Result Access

- `ListOfTables()` - Get list of available tables
- `getTable(name)` - Get specific table by name
- `getTableIndex()` - Get table index with metadata

Result Display

- `printResults()` - Print all results to console
- `showResults(name, options)` - Show specific table
- `showTableIndex(options)` - Show table directory
- `showGraph(tableName)` - Display associated graphs

Result Export

- `exportResults(format)` - Export all tables in specified format
- `exportTable(tableName, format)` - Export specific table
- `saveResults(filename)` - Save all results to file
- `saveTable(tableName, filename)` - Save specific table

Base Functions for cResultSet Objects

Base functions provide a unified interface for working with any cResultSet object, including cThermoeconomicModel, cResultInfo, and cDataModel.

Display Functions

`ShowResults(resultSet, Name, Value)`

Purpose: Universal result display interface
Features: Console, GUI, HTML display; table selection; optional saving

Parameters:

- `Table` - Specific table name (optional)
- `View` - Display mode: 'CONSOLE', 'GUI', 'HTML'
- `Panel` - Use interactive ResultsPanel
- `SaveAs` - Save to file simultaneously

Examples:

```
% Show all results in console
ShowResults(model);

% Show specific table in HTML
ShowResults(model, 'Table', 'dcost', 'View', 'HTML');

% Show with interactive panel
ShowResults(model, 'Panel', true);

% Show and save simultaneously
ShowResults(model, 'Table', 'efficiency', 'SaveAs', 'exergy_results.xlsx');
```

`ShowTable(table, Name, Value)`

Purpose: Display individual cTable objects
Features: Multiple view modes, export options

Example:

```
table = model.getTable('dcost');
ShowTable(table, 'View', 'HTML');
```

`ShowGraph(resultSet, tableName)`

Purpose: Display graphs associated with result tables
Example:

```
ShowGraph(model, 'cost_formation');
```

`ListResultTables(resultSet, Name, Value)`

Purpose: Display table directory with metadata
Features: Customizable columns, multiple display formats

Example:

```
% List all available tables
ListResultTables(model);

% Custom column selection
ListResultTables(model, 'Columns', {'DESCRIPTION', 'TYPE', 'GRAPH'});
```

Export Functions

`ExportResults(resultSet, Name, Value)`

Purpose: Export results in various formats
Formats: CELL, STRUCT, TABLE, NONE (cTable objects)

Parameters:

- `Table` - Specific table (optional, exports all if omitted)
- `ExportAs` - Output format
- `Format` - Apply number formatting

Examples:

```
% Export all tables as MATLAB tables
allTables = ExportResults(model, 'ExportAs', 'TABLE');

% Export specific table as cell array
costTable = ExportResults(model, 'Table', 'dcost', 'ExportAs', 'CELL');

% Export with formatting applied
formattedResults = ExportResults(model, 'ExportAs', 'STRUCT', 'Format', true);
```

Save Functions

`SaveResults(resultSet, filename)`

Purpose: Save all result tables to file
Formats: XLSX, CSV, HTML, TXT, LaTeX, MAT

Examples:

```
% Save to Excel
SaveResults(model, 'thermoeconomic_analysis.xlsx');

% Save to CSV (creates directory with individual files)
SaveResults(model, 'results.csv');

% Save to HTML
SaveResults(model, 'report.html');
```

`SaveTable(table, filename)`

Purpose: Save individual table to file
Example:

```
costTable = model.getTable('dcost');
SaveTable(costTable, 'cost_analysis.xlsx');
```

Interactive Functions

`ViewResults(resultSet)`

Purpose: Launch interactive MATLAB App for result exploration
Features: Tree navigation, table/graph display, save functionality

Example:

```
% Launch interactive viewer
ViewResults(model);
```

`ResultsPanel(resultSet)`

Purpose: Create interactive results panel
Features: Table selection, multiple view modes, save options

Example:

```
% Create and show results panel
panel = ResultsPanel(model);
```

Usage Examples

Complete Analysis Workflow

```
% 1. Create model with full configuration
model = ThermoeconomicModel('cogeneration_plant.json', ...
    'State', 'design', ...
    'ResourceSample', 'base_costs', ...
    'CostTables', 'ALL', ...
    'Summary', 'STATES', ...
    'Debug', false);

% 2. Verify model capabilities
fprintf('Resource costs available: %s\n', mat2str(model.isResourceCost()));
fprintf('Diagnosis possible: %s\n', mat2str(model.isDiagnosis()));
fprintf('Waste analysis available: %s\n', mat2str(model.isWaste()));

% 3. Perform sequential analysis
structure = model.productiveStructure();
exergy = model.exergyAnalysis();
thermo = model.thermoeconomicAnalysis();

% 4. Display key results
ShowResults(structure, 'Table', 'flows');
ShowResults(exergy, 'Table', 'efficiency', 'View', 'HTML');
ShowResults(thermo, 'Table', 'dcost', 'View', 'HTML');

% 5. Save comprehensive results
SaveResults(thermo, 'complete_analysis.xlsx');
```

Interactive Analysis

```
% 1. Create model
model = ThermoeconomicModel('plant_model.json');

% 2. Launch interactive interface
ViewResults(model);

% Alternative: Use results panel
ShowResults(model, 'Panel', true);
```

Comparative Analysis

```
% 1. Create model with multiple states
model = ThermoeconomicModel('multi_state_model.json', ...
    'Summary', 'STATES');

% 2. Analyze design state
model.setState('design');
designResults = model.thermoeconomicAnalysis();

% 3. Analyze off-design state
model.setState('off_design');
offDesignResults = model.thermoeconomicAnalysis();

% 4. Perform diagnosis
if model.isDiagnosis()
    diagnosis = model.thermoeconomicDiagnosis();
    ShowResults(diagnosis);
end

% 5. Generate summary
summary = model.summaryResults();
ShowResults(summary, 'View', 'HTML');
```

Parametric Study

```
% 1. Base model setup
baseModel = ThermoeconomicModel('parametric_base.json');

% 2. Define parameter variations
samples = {'case1_costs', 'case2_costs', 'case3_costs'};
results = cell(length(samples), 1);

% 3. Analyze each case
for i = 1:length(samples)
    baseModel.setResourceSample(samples{i});
    results{i} = baseModel.thermoeconomicAnalysis();

    % Save individual results
    filename = sprintf('case_%d_results.xlsx', i);
    SaveResults(results{i}, filename);
end

% 4. Compare results
fprintf('Parametric study completed for %d cases\n', length(samples));
```

Advanced Configuration

```
% 1. Create model with waste analysis
model = ThermoeconomicModel('waste_model.json', ...
    'ActiveWaste', 'QC', ...
    'Recycling', true, ...
    'DiagnosisMethod', 'WASTE_INTERNAL');

% 2. Comprehensive analysis
if model.isWaste()
    wasteResults = model.wasteAnalysis();
    ShowResults(wasteResults, 'Table', 'waste_allocation');
end

% 3. Generate diagrams
diagrams = model.productiveDiagram();
fpDiagrams = model.diagramFP();

% 4. Export for external analysis
allData = ExportResults(model, 'ExportAs', 'STRUCT');
save('model_data.mat', 'allData');
```

Common Patterns

Standard Analysis Pattern

```
% 1. Create → 2. Configure → 3. Analyze → 4. Present → 5. Save
model = ThermoeconomicModel(dataSource);
model.setState('operating_point');
results = model.thermoeconomicAnalysis();
ShowResults(results, 'View', 'HTML');
SaveResults(results, 'analysis.xlsx');
```

Validation Pattern

```
% Always check capabilities before analysis
if model.isResourceCost()
    thermoResults = model.thermoeconomicAnalysis();
end

if model.isDiagnosis()
    diagnosis = model.thermoeconomicDiagnosis();
end

if model.isWaste()
    wasteResults = model.wasteAnalysis();
end
```

Error Handling Pattern

```
try
    model = ThermoeconomicModel('data_file.json');
    if model.status
        results = model.thermoeconomicAnalysis();
        ShowResults(results);
    else
        model.printLogger();
    end
catch ME
    fprintf('Error: %s\n', ME.message);
end
```

Interactive Pattern

```
% For GUI applications and interactive analysis
model = ThermoeconomicModel(dataFile);
ViewResults(model);  % Launches full interactive interface

% Or use results panel for specific workflow
ShowResults(model, 'Panel', true);
```

See Also

Related TaesLab Components

- ReadDataModel Guide - Data model creation and management
- cDataModel Class - Data model container
- cResultSet Class - Base class for result management
- cResultInfo Class - Result organization and presentation

Base Functions

- ShowResults - Universal result display
- SaveResults - Universal result saving
- ExportResults - Result format conversion
- ViewResults - Interactive result viewer

Analysis Functions

- ExergyAnalysis - Direct exergy analysis
- ThermoeconomicAnalysis - Direct thermoeconomic analysis
- ThermoeconomicDiagnosis - Direct diagnosis analysis
- WasteAnalysis - Direct waste analysis

Example Models

- Examples Directory - Complete plant models
- Rankine Cycle - Basic thermoeconomic model
- Cogeneration - Multi-state analysis example
- Combined Cycle - Complex system example

Generated: November 2025
TaesLab Version: 1.8