Reference Guide: ReadDataModel and cDataModel

This guide provides complete documentation for the `ReadDataModel.m` function, the `cDataModel` class, the `cReadModel` interface, and base functions that work with data models in TaesLab.

Table of Contents

ReadDataModel Function

The `ReadDataModel` function is the main function for loading data model files in TaesLab.

Syntax

```
data = ReadDataModel(filename, Name, Value)
```

Input Arguments

`filename` (required)

- **Type**: `char array | string`
- **Description**: Name of the file with the data model
- **Supported formats**: JSON, XML, XLSX, CSV, MAT

Name-Value Arguments

Parameter	Type	Description
`Debug`	`logical`	Show detailed validation in console (default: false)
`Show`	`logical`	Show data tables in console (default: false)
`SaveAs`	`char/string`	Filename to save a copy of the model

Return Value

- **Type**: `cDataModel`
- **Description**: Validated data model object ready for analysis

Examples

```
% Load basic model
data = ReadDataModel('rankine_model.json');

% Load with debug information
data = ReadDataModel('plant_model.xlsx', 'Debug', true);

% Load and show tables
data = ReadDataModel('model.xml', 'Show', true, 'Debug', true);

% Load and save copy to MAT
data = ReadDataModel('model.csv', 'SaveAs', 'model_copy.mat');
```

Validation and Verification

The function performs the following validations:

- **File existence**: Verifies that the file exists
- **Valid format**: Checks file extension and format
- **Model structure**: Validates productive structure, flows and processes
- **Exergy data**: Verifies consistency of thermodynamic states
- **Cost data**: Validates resource cost information (if exists)
- **Waste definition**: Verifies waste configuration (if exists)

cDataModel Class

The `cDataModel` class is the central container for validated data in TaesLab, derived from `cResultSet` .

Properties

Read-Only Properties - System Information

Property	Type	Description
`NrOfFlows`	`double`	Number of system flows
`NrOfProcesses`	`double`	Number of processes
`NrOfWastes`	`double`	Number of waste flows
`NrOfResources`	`double`	Number of resources
`NrOfSystemOutputs`	`double`	Number of system outputs
`NrOfFinalProducts`	`double`	Number of final products
`NrOfStates`	`double`	Number of thermodynamic states
`NrOfSamples`	`double`	Number of cost samples

Read-Only Properties - Capability Indicators

Property	Type	Description
`isWaste`	`logical`	Indicates if the model has defined wastes
`isResourceCost`	`logical`	Indicates if it has resource cost data
`isDiagnosis`	`logical`	Indicates if it can perform diagnosis
`isSummary`	`logical`	Indicates if it can generate summary reports

Read-Only Properties - Names and Labels

Property	Type	Description
`StateNames`	`cell array`	Names of thermodynamic states
`SampleNames`	`cell array`	Names of cost samples
`WasteFlows`	`cell array`	Names of waste flows

Read-Only Properties - Data Objects

Property	Type	Description
`ProductiveStructure`	`cProductiveStructure`	Productive structure of the system
`FormatData`	`cResultTableBuilder`	Result table builder
`WasteData`	`cWasteData`	Waste and recycling data
`ExergyData`	`cDataset`	Exergy data set
`ResourceData`	`cDataset`	Cost data set
`SummaryOptions`	`cSummaryOptions`	Summary report options
`ModelData`	`cModelData`	Model data from cReadModel

Data Access Methods

Thermodynamic State Methods

`existState(obj, stateName)`

Purpose: Verify if a thermodynamic state exists
Syntax: `res = obj.existState(stateName)`
Input: `stateName` - State name (char/string)
Return: `logical` - true if state exists
Example:

```
if data.existState('design')
    exergyData = data.getExergyData('design');
end
```

`getExergyData(obj, stateName)`

Purpose: Get exergy data for a specific state
Syntax: `exergyData = obj.getExergyData(stateName)`
Input: `stateName` - State name
Return: `cExergyData` - Object with exergy data
Example:

```
designData = data.getExergyData('design');
offDesignData = data.getExergyData('off_design');
```

Cost Sample Methods

`existSample(obj, sampleName)`

Purpose: Verify if a cost sample exists
Syntax: `res = obj.existSample(sampleName)`
Input: `sampleName` - Sample name
Return: `logical` - true if sample exists
Example:

```
if data.existSample('summer_costs')
    resourceData = data.getResourceData('summer_costs');
end
```

`getResourceData(obj, sampleName)`

Purpose: Get resource data for a specific sample
Syntax: `resourceData = obj.getResourceData(sampleName)`
Input: `sampleName` - Sample name
Return: `cResourceData` - Object with cost data
Example:

```matlab
summerCosts = data.getResourceData('summer_costs');
winterCosts = data.getResourceData('winter_costs');
```

Waste Methods

`getWasteDefinition(obj)`

Purpose: Get waste definition information
Syntax: `wasteInfo = obj.getWasteDefinition()`
Return: Structure with waste information
Example:

```matlab
if data.isWaste
    wasteInfo = data.getWasteDefinition();
    disp(wasteInfo);
end
```

Modification Methods

Exergy State Modification

`setExergyData(obj, stateName, exergyData)`

Purpose: Set exergy values for a state
Syntax: `obj.setExergyData(stateName, exergyData)`
Input:

- `stateName` - State name
- `exergyData` - cExergyData object

`addExergyData(obj, stateName, exergyData)`

Purpose: Add new exergy state
Syntax: `obj.addExergyData(stateName, exergyData)`
Input:

- `stateName` - New state name
- `exergyData` - Exergy data

Example:

```matlab
% Modify existing state
newExergyData = cExergyData(...);
data.setExergyData('design', newExergyData);

% Add new state
data.addExergyData('part_load', partLoadData);
```

Resource Modification

`setFlowResource(obj, sampleName, flowValues)`

Purpose: Set resource values for flows
Syntax: `obj.setFlowResource(sampleName, flowValues)`
Input:

- `sampleName` - Sample name
- `flowValues` - Cost values for flows

`setProcessResource(obj, sampleName, processValues)`

Purpose: Set resource values for processes
Syntax: `obj.setProcessResource(sampleName, processValues)`
Input:

- `sampleName` - Sample name
- `processValues` - Cost values for processes

`addResourceData(obj, sampleName, resourceData)`

Purpose: Add new resource sample
Syntax: `obj.addResourceData(sampleName, resourceData)`
Input:

- `sampleName` - New sample name
- `resourceData` - Resource data

Example:

```
% Modify flow costs
flowCosts = [10.5, 15.2, 8.0]; % Example costs
data.setFlowResource('summer_costs', flowCosts);

% Add new cost sample
newResourceData = cResourceData(...);
data.addResourceData('autumn_costs', newResourceData);
```

Waste Modification

`setWasteType(obj, wasteName, wasteType)`

Purpose: Modify the type of a waste flow
Syntax: `obj.setWasteType(wasteName, wasteType)`
Input:

- `wasteName` - Waste flow name
- `wasteType` - Waste type

`setWasteValues(obj, wasteName, allocationValues)`

Purpose: Modify allocation values for a waste flow
Syntax: `obj.setWasteValues(wasteName, allocationValues)`
Input:

- `wasteName` - Waste name
- `allocationValues` - Allocation values

`setWasteRecycled(obj, wasteName, recyclingRatio)`

Purpose: Modify recycling ratio for a waste flow
Syntax: `obj.setWasteRecycled(wasteName, recyclingRatio)`
Input:

- `wasteName` - Waste name
- `recyclingRatio` - Recycling ratio (0-1)

Example:

```
% Modify waste type
data.setWasteType('QC', cType.WasteType.INTERNAL);

% Set allocation values
allocationValues = [0.3, 0.4, 0.3]; % Distribution among processes
data.setWasteValues('QC', allocationValues);

% Set recycling ratio
data.setWasteRecycled('QC', 0.15); % 15% recycling
```

Information Methods

`getTablesDirectory(obj, cols)`

Purpose: Get directory of available tables
Syntax: `directory = obj.getTablesDirectory(cols)`
Input: `cols` - Columns to show (optional)
Return: Table with directory of tables
Example:

```
directory = data.getTablesDirectory();
disp(directory);
```

`getTableInfo(obj, tableName)`

Purpose: Get information about a specific table
Syntax: `info = obj.getTableInfo(tableName)`
Input: `tableName` - Table name
Return: Structure with table information
Example:

```
flowsInfo = data.getTableInfo('flows');
disp(flowsInfo);
```

`getResultInfo(obj)`

Purpose: Get cResultInfo associated with the data model
Syntax: `resultInfo = obj.getResultInfo()`
Return: `cResultInfo` with model tables
Example:

```
dataInfo = data.getResultInfo();
ShowResults(dataInfo);
```

`showDataModel(obj)`

Purpose: Display the data model
Syntax: `obj.showDataModel()`
Description: Shows all model tables in console
Example:

```
data.showDataModel();
```

`saveDataModel(obj, filename)`

Purpose: Save the data model
Syntax: `obj.saveDataModel(filename)`
Input: `filename` - Output file name
Example:

```
data.saveDataModel('updated_model.json');
data.saveDataModel('model_backup.mat');
```

cReadModel Interface

The `cReadModel` interface provides unified reading of different file formats.

Class Hierarchy

```
cReadModel (Abstract)
├── cReadModelStruct (Abstract)
│   ├── cReadModelJSON (JSON files)
│   └── cReadModelXML (XML files)
└── cReadModelTable (Abstract)
    ├── cReadModelXLS (Excel files)
    └── cReadModelCSV (CSV files)
```

Format Implementations

`cReadModelJSON`

Purpose: Read JSON files with data structure
Features:

- Direct parsing with `jsondecode`
- Complete support for nested structures
- JSON schema validation

`cReadModelXML`

Purpose: Read XML files with JSON conversion
Features:

- XML→JSON conversion for uniform processing
- Support for XML attributes and elements
- XML schema validation

`cReadModelXLS`

Purpose: Read multi-sheet Excel files
Features:

- Multi-worksheet reading
- Configuration based on `printformat.json`
- Support for tabular data and matrices

`cReadModelCSV`

Purpose: Read multiple CSV files in directory
Features:

- Reading of related CSV files
- Expected file name configuration
- Tabular data processing

Supported Formats

JSON Format

Typical structure:

```
{
  "ProductiveStructure": {
    "flows": [
      {"key": "CMB", "type": "RESOURCE"},
      {"key": "WN", "type": "OUTPUT"}
    ],
    "processes": [
      {"key": "BOIL", "fuel": "CMB", "product": "B1+B2"}
    ]
  },
  "ExergyStates": {
    "design": {
      "flows": [10.5, 15.2, 8.0],
      "processes": [5.2, 3.1]
    }
  },
  "ResourcesCost": {
    "summer": {
      "flows": [0.05, 0.08],
      "processes": [100, 150]
    }
  }
}
```

XML Format

Typical structure:

```
<TaesLabModel>
  <ProductiveStructure>
    <flows>
      <flow key="CMB" type="RESOURCE"/>
    </flows>
    <processes>
      <process key="BOIL" fuel="CMB" product="B1+B2"/>
    </processes>
  </ProductiveStructure>
</TaesLabModel>
```

Excel Format (XLSX)

Required sheets:

- `Flows` : Flow definitions
- `Processes` : Process definitions
- `ExergyStates` : Thermodynamic states
- `ResourcesCost` : Resource costs (optional)
- `WasteDefinition` : Waste definitions (optional)

CSV Format

Required files:

- `Flows.csv` : System flows
- `Processes.csv` : Productive processes
- `ExergyStates.csv` : Exergy states
- `ResourcesCost.csv` : Costs (optional)

Base Functions for cDataModel

`SaveDataModel(dataModel, filename)`

Purpose: Save data model to file
Syntax: `SaveDataModel(dataModel, filename)`
Input:

- `dataModel` - cDataModel or cThermoeconomicModel object
- `filename` - File name with extension

Supported formats: XLSX, CSV, JSON, XML, MAT
Example:

```
SaveDataModel(data, 'updated_model.json');
SaveDataModel(data, 'model_tables.xlsx');
SaveDataModel(data, 'backup.mat');
```

`ImportDataModel(filename)`

Purpose: Import data model from MAT file
Syntax: `dataModel = ImportDataModel(filename)`
Input: `filename` - MAT file with TaesLab object
Return: `cDataModel` - Data model object
Example:

```
% Import previously saved model
savedData = ImportDataModel('previous_model.mat');
if isValid(savedData)
    % Use imported model
    model = ThermoeconomicModel(savedData);
end
```

Analysis Functions Using cDataModel

`ExergyAnalysis(dataModel, Name, Value)`

Purpose: Perform direct exergy analysis
Syntax: `results = ExergyAnalysis(dataModel, Name, Value)`

Example:

```
exergyResults = ExergyAnalysis(data, 'State', 'design');
ShowResults(exergyResults);
```

`ThermoeconomicAnalysis(dataModel, Name, Value)`

Purpose: Perform direct thermoeconomic analysis
Syntax: `results = ThermoeconomicAnalysis(dataModel, Name, Value)`
Example:

```
thermoResults = ThermoeconomicAnalysis(data, ...
    'State', 'design', ...
    'ResourceSample', 'summer_costs');
```

`ThermoeconomicDiagnosis(dataModel, Name, Value)`

Purpose: Perform direct thermoeconomic diagnosis
Syntax: `results = ThermoeconomicDiagnosis(dataModel, Name, Value)`
Example:

```
diagnosisResults = ThermoeconomicDiagnosis(data, ...
    'State', 'off_design', ...
    'ReferenceState', 'design');
```

`WasteAnalysis(dataModel, Name, Value)`

Purpose: Perform direct waste analysis
Syntax: `results = WasteAnalysis(dataModel, Name, Value)`
Example:

```
wasteResults = WasteAnalysis(data, ...
    'ActiveWaste', 'QC', ...
    'Recycling', true);
```

`SummaryResults(dataModel, Name, Value)`

Purpose: Generate summary results
Syntax: `results = SummaryResults(dataModel, Name, Value)`
Example:

```
summaryResults = SummaryResults(data, 'Summary', 'STATES');
ShowResults(summaryResults);
```

Usage Examples

Load and Validate Basic Model

```matlab
% 1. Load model with validation
data = ReadDataModel('rankine_model.json', 'Debug', true);

% 2. Check successful loading
if isValid(data)
    fprintf('Model loaded successfully\n');
    fprintf('Flows: %d, Processes: %d\n', data.NrOfFlows, data.NrOfProcesses);
else
    fprintf('Error loading model\n');
    return;
end

% 3. Show model information
data.showDataModel();

% 4. Check capabilities
fprintf('Resource costs: %s\n', mat2str(data.isResourceCost));
fprintf('Diagnosis available: %s\n', mat2str(data.isDiagnosis));
fprintf('Waste defined: %s\n', mat2str(data.isWaste));
```

Explore Model Structure

```matlab
% Load model
data = ReadDataModel('plant_model.xlsx');

% Explore available states
fprintf('Available states: %s\n', strjoin(data.StateNames, ', '));

% Explore cost samples
if data.isResourceCost
    fprintf('Cost samples: %s\n', strjoin(data.SampleNames, ', '));
end

% Explore waste flows
if data.isWaste
    fprintf('Waste flows: %s\n', strjoin(data.WasteFlows, ', '));
end

% Get tables directory
directory = data.getTablesDirectory();
disp(directory);
```

Modify Model Data

```
% Load model
data = ReadDataModel('model.json');

% Check existing states
if data.existState('design')
    designData = data.getExergyData('design');
    % Modify exergy data if needed
end

% Check cost samples
if data.existSample('base_costs')
    costData = data.getResourceData('base_costs');
    % Modify costs if needed
end

% Modify waste configuration
if data.isWaste
    data.setWasteType('QC', cType.WasteType.INTERNAL);
    data.setWasteRecycled('QC', 0.10); % 10% recycling
end

% Save modified model
SaveDataModel(data, 'modified_model.json');
```

Format Conversion

```
% Load Excel model
data = ReadDataModel('model.xlsx', 'Debug', true);

% Save in different formats
SaveDataModel(data, 'model.json');    % JSON
SaveDataModel(data, 'model.xml');     % XML
SaveDataModel(data, 'model.mat');     % MAT
SaveDataModel(data, 'model_csv');     % CSV (directory)

fprintf('Model converted to multiple formats\n');
```

Direct Analysis with cDataModel

```
% Load model
data = ReadDataModel('cogeneration_model.json');

% Direct exergy analysis
exergyResults = ExergyAnalysis(data, 'State', 'design');
ShowResults(exergyResults, 'Table', 'efficiency');

% Direct thermoeconomic analysis
if data.isResourceCost
    thermoResults = ThermoeconomicAnalysis(data, ...
        'State', 'design', ...
        'ResourceSample', 'base_costs');
    ShowResults(thermoResults, 'Table', 'dcost');
end

% Diagnosis if multiple states available
if data.isDiagnosis
    diagnosisResults = ThermoeconomicDiagnosis(data, ...
        'State', 'off_design', ...
        'ReferenceState', 'design');
    ShowResults(diagnosisResults);
end
```

Common Usage Patterns

1. Model Validation and Debugging

```matlab
% Load with detailed information
data = ReadDataModel('new_model.json', 'Debug', true, 'Show', true);

% Check structure
if ~isValid(data)
    fprintf('Model errors:\n');
    data.printLogger();
    return;
end

% Check specific tables
flowsInfo = data.getTableInfo('flows');
processesInfo = data.getTableInfo('processes');

% Show model summary
fprintf('=== MODEL SUMMARY ===\n');
fprintf('Flows: %d\n', data.NrOfFlows);
fprintf('Processes: %d\n', data.NrOfProcesses);
fprintf('States: %d\n', data.NrOfStates);
fprintf('Cost samples: %d\n', data.NrOfSamples);
```

2. Format Comparison

```matlab
% Load same model in different formats
dataJSON = ReadDataModel('model.json');
dataXLSX = ReadDataModel('model.xlsx');
dataXML  = ReadDataModel('model.xml');

% Check consistency
formats = {'JSON', 'XLSX', 'XML'};
models = {dataJSON, dataXLSX, dataXML};

for i = 1:length(models)
    data = models{i};
    if isValid(data)
        fprintf('%s: %d flows, %d processes\n', ...
            formats{i}, data.NrOfFlows, data.NrOfProcesses);
    else
        fprintf('%s: Loading error\n', formats{i});
    end
end
```

3. Model Migration and Updates

```
% Load old model
oldData = ImportDataModel('old_model.mat');

if isValid(oldData)
    % Add new state
    newStateData = cExergyData(...); % Configure new state
    oldData.addExergyData('new_operating_point', newStateData);

    % Add new cost sample
    newCostData = cResourceData(...); % Configure new costs
    oldData.addResourceData('updated_costs', newCostData);

    % Save updated model
    SaveDataModel(oldData, 'updated_model.json');

    fprintf('Model updated and saved\n');
end
```

4. Parametric Analysis

```
% Load base model
baseData = ReadDataModel('base_model.json');

% List of parametric variations
variations = {'case1.json', 'case2.json', 'case3.json'};

results = cell(length(variations), 1);

for i = 1:length(variations)
    % Load variation
    data = ReadDataModel(variations{i});

    if isValid(data)
        % Perform analysis
        result = ExergyAnalysis(data, 'State', 'design');
        results{i} = result;

        % Save individual results
        filename = sprintf('results_case%d.xlsx', i);
        SaveResults(result, filename);
    end
end

fprintf('Parametric analysis completed\n');
```

5. Processing Automation

```
function processModelsInDirectory(directory)
    % Process all models in a directory

    files = dir(fullfile(directory, '*.json'));

    for i = 1:length(files)
        filename = fullfile(directory, files(i).name);
        fprintf('Processing: %s\n', files(i).name);

        try
            % Load model
            data = ReadDataModel(filename, 'Debug', false);

            if isValid(data)
                % Create thermoeconomic model
                model = ThermoeconomicModel(data, 'CostTables', 'ALL');

                % Perform analysis
                results = model.thermoeconomicAnalysis();

                % Save results
                [~, name, ~] = fileparts(files(i).name);
                outputFile = fullfile(directory, [name '_results.xlsx']);
                SaveResults(results, outputFile);

                fprintf('  ✓ Completed\n');
            else
                fprintf('  ✗ Model error\n');
            end

        catch ME
            fprintf('  ✗ Error: %s\n', ME.message);
        end
    end
end

% Use the function
processModelsInDirectory('models_directory');
```

See Also

- ThermoeconomicModel Guide
- TaesLab Classes Reference
- Model Examples
- Utility Functions

Generated: November 2025

TaesLab Version: 1.8