TaesLab Base Functions Analysis

Analysis Date: November 2, 2025

Overview

The Base Functions in TaesLab serve as the **user interface layer** between end users and the complex class hierarchy. These functions provide a simplified, consistent API that handles parameter validation, error management, and result presentation while leveraging the sophisticated object-oriented infrastructure underneath.

1. Base Functions Architecture

1.1 Function Categories

Based on the Contents.m file, Base Functions are organized into six main categories:

Apps

- `TaesApp` - Full MATLAB App Designer application
- `TaesTool` - Command-line compatible interface for MATLAB/Octave

Read Data Models

- `ValidateModelTables` - Validate table data model files
- `ReadDataModel` - Read a data model file (universal format support)
- `CopyDataModel` - Create copies in different formats
- `ImportDataModel` - Load from previously saved MAT files
- `ImportData` - Import external data from CSV/XLSX

Get Thermoeconomic Results

- `ThermoeconomicModel` - Create cThermoeconomicModel object
- `ProductiveStructure` - Get plant productive structure
- `ProductiveDiagram` - Generate productive diagrams
- `ExergyAnalysis` - Perform exergy analysis
- `ThermoeconomicAnalysis` - Perform thermoeconomic analysis
- `ThermoeconomicDiagnosis` - Compare plant states
- `WasteAnalysis` - Waste recycling analysis
- `DiagramFP` - Generate FP diagrams
- `SummaryResults` - Generate summary reports

Save Results Tables

- `SaveResults` - Save results to files
- `SaveSummary` - Save summary results
- `SaveDataModel` - Save data model tables
- `SaveTable` - Save individual tables

Display Results

- `ListResultTables` - List available tables and properties
- `ShowResults` - Display results in different formats
- `ShowTable` - Display individual tables
- `ShowGraph` - Display graphs from result tables
- `ExportResults` - Export results in different formats

GUI Functions

- `TaesPanel` - Parameter selection GUI
- `ResultsPanel` - Interactive results display GUI
- `ViewResults` - MATLAB app for table viewing

2. Class Usage Patterns in Base Functions

2.1 Standard Base Function Architecture

All Base Functions follow a consistent architectural pattern:

```
function res = BaseFunction(data, varargin)
    % 1. Initialize logger
    res = cTaesLab();

    % 2. Input validation
    if nargin < 1 || ~isObject(data, 'cDataModel')
        res.printError(cMessages.DataModelRequired, cMessages.ShowHelp);
        return
    end

    % 3. Parameter parsing
    p = inputParser;
    p.addParameter('Show', false, @islogical);
    p.addParameter('SaveAs', cType.EMPTY_CHAR, @isFilename);
    try
        p.parse(varargin{:});
    catch err
        res.printError(err.message);
        return
    end

    % 4. Create computation objects
    computationObject = cAnalysisClass(data);

    % 5. Error handling and result generation
    if computationObject.status
        res = computationObject.buildResultInfo(data.FormatData);
    else
        computationObject.printLogger;
        res.printError(cMessages.InvalidObject, class(computationObject));
        return
    end

    % 6. Optional display/save operations
    if param.Show
        printResults(res);
    end
    if ~isEmpty(param.SaveAs)
        SaveResults(res, param.SaveAs);
    end
end
```

2.2 Core Classes Used by Base Functions

Universal Base Classes

- `cTaesLab` - Used as logger object in every Base Function
- `cType` - Constants and validation methods used throughout
- `cMessages` - Standardized error messages

Data Management Classes

- **`cDataModel`** - Primary input for most analysis functions
- **`cReadModel`** - File reading interface (via `ReadDataModel`)
- **`cModelData`** - Internal data container
- **`cResultTableBuilder`** - Formatting and table generation

Analysis Classes

- **`cExergyModel`** - Used by `ExergyAnalysis`
- **`cExergyCost`** - Used by `ThermoeconomicAnalysis`
- **`cDiagnosis`** - Used by `ThermoeconomicDiagnosis`
- **`cWasteAnalysis`** - Used by `WasteAnalysis`
- **`cProductiveStructure`** - Used by `ProductiveStructure`
- **`cSummaryResults`** - Used by `SummaryResults`

Result Management Classes

- **`cResultInfo`** - Standard output for all analysis functions
- **`cResultSet`** - Base for result presentation
- **`cThermoeconomicModel`** - Comprehensive analysis engine

Table and Display Classes

- **`cTable`** variants - Table representation and formatting
- **`cGraph`** variants - Graphical presentation

3. Detailed Function Analysis

3.1 Data Input Functions

ReadDataModel

```
data = ReadDataModel(filename, 'Debug', true, 'Show', false)
```

Classes Used:

- `cTaesLab` (error logging)
- `cDataModel.create()` (factory method)
- File reading via `cReadModel` interface

Purpose: Universal file reader with automatic format detection (JSON, XML, XLSX, CSV, MAT)

ThermoeconomicModel

```
model = ThermoeconomicModel(data, 'CostTables', 'ALL', 'Summary', 'STATES')
```

Classes Used:

- `cThermoeconomicModel` (main analysis engine)
- `cDataModel` (data source)

- `cModelResults` (result container)

Purpose: Creates comprehensive analysis engine object

3.2 Analysis Functions

ExergyAnalysis

```
res = ExergyAnalysis(data, 'State', 'design', 'Show', true)
```

Classes Used:

- `cTaesLab` (error logging)
- `cDataModel` (input validation and data access)
- `cExergyData` (thermodynamic data extraction)
- `cExergyModel` (computation engine)
- `cResultInfo` (result packaging)

Data Flow:

```
cDataModel → getExergyData() → cExergyData → cExergyModel → buildResultInfo() →
cResultInfo
```

ThermoeconomicAnalysis

```
res = ThermoeconomicAnalysis(data, 'CostTables', 'ALL', 'ResourceSample', 'summer')
```

Classes Used:

- `cTaesLab` (error messages)
- `cDataModel` (input and resource data)
- `cExergyData` (thermodynamic foundation)
- `cExergyCost` (cost computation)
- `cResultInfo` (result packaging)

Data Flow:

```
cDataModel → getExergyData() + getResourceData() → cExergyCost → buildResultInfo() →
cResultInfo
```

ThermoeconomicDiagnosis

```
res = ThermoeconomicDiagnosis(data, 'State', 'off_design', 'DiagnosisMethod',
'WASTE_INTERNAL')
```

Classes Used:

- `cTaesLab` (error logging)
- `cDataModel` (multi-state data access)

- `cExergyData` (reference and comparison states)
- `cDiagnosis` (comparative analysis)
- `cResultInfo` (diagnosis results)

Data Flow:

```
cDataModel → getExergyData(ref) + getExergyData(state) → cDiagnosis → buildResultInfo() →
cResultInfo
```

3.3 Result Management Functions

ShowResults

```
ShowResults(results, 'Table', 'dcost', 'View', 'HTML')
```

Classes Used:

- `cTaesLab` (error logging)
- `cResultSet` (input validation)
- `cTable` variants (table display)
- `cTableIndex` (table navigation)

Purpose: Unified result display interface

SaveResults

```
SaveResults(results, 'analysis_results.xlsx')
```

Classes Used:

- `cTaesLab` (error messages)
- `cResultSet` (result access)
- `cTable` export methods
- Various export utilities (`exportXLS` , `exportCSV` , etc.)

4. Error Handling and Logging

4.1 cMessageLogger Integration

Base Functions serve as the **error interface layer** using `cMessageLogger` architecture:

1. **Immediate Validation**: Base Functions validate inputs and display errors immediately
2. **Error Accumulation**: Object classes accumulate detailed errors in message loggers
3. **Centralized Display**: Base Functions call `printLogger()` to show accumulated errors
4. **Status Propagation**: Object `status` properties indicate validity throughout the chain

4.2 Error Handling Pattern

```
% Create computation object (accumulates errors)
analysisObject = cAnalysisClass(data);

% Check status and display errors if needed
if analysisObject.status
    res = analysisObject.buildResultInfo(formatData);
else
    analysisObject.printLogger;  % Display accumulated errors
    res.printError(cMessages.InvalidObject, class(analysisObject));
    return
end

% Final validation
if ~res.status
    res.printLogger;
    return
end
```

5. Parameter Validation and Type Checking

5.1 Validation Functions Used

Base Functions leverage extensive validation utilities:

- **`isObject(obj, 'className')`** - Class type validation
- **`isFilename(filename)`** - File name validation
- **`data.existState(state)`** - State name validation
- **`cType.check*()`** methods - Enum and constant validation
- **`@islogical`**, **`@ischar`** - MATLAB built-in validators

5.2 Parameter Processing Pattern

```
p = inputParser;
p.addParameter('State', data.StateNames{1}, @data.existState);
p.addParameter('Show', false, @islogical);
p.addParameter('CostTables', cType.DEFAULT_COST_TABLES, @cType.checkCostTables);
try
    p.parse(varargin{:});
catch err
    res.printError(err.message);
    return
end
param = p.Results;
```

6. Result Generation and Presentation

6.1 buildResultInfo Pattern

All analysis functions use the standard `buildResultInfo()` pattern:

```
% Analysis object creates results
analysisObject = cAnalysisClass(inputData);

% Generate formatted results using FormatData
res = analysisObject.buildResultInfo(data.FormatData);
```

6.2 Optional Presentation

Base Functions provide consistent optional presentation:

```
% Display results if requested
if param.Show
    printResults(res);  % or res.printResults()
end

% Save results if filename provided
if ~isempty(param.SaveAs)
    SaveResults(res, param.SaveAs);
end
```

7. Class Dependency Summary

7.1 Most Used Classes in Base Functions

1. **`cTaesLab`** - Universal error logger (used in every function)
2. **`cDataModel`** - Primary data source (used in all analysis functions)
3. **`cResultInfo`** - Standard result container (output of all analyses)
4. **`cType`** - Constants and validation (used throughout)
5. **`cMessages`** - Error messages (used throughout)

7.2 Analysis-Specific Classes

- **Exergy Analysis**: `cExergyData`, `cExergyModel`
- **Cost Analysis**: `cExergyCost`, `cResourceData`
- **Diagnosis**: `cDiagnosis`
- **Waste Analysis**: `cWasteAnalysis`, `cWasteData`
- **Structure Analysis**: `cProductiveStructure`
- **Summary**: `cSummaryResults`, `cSummaryOptions`

7.3 Utility Classes

- **File Operations**: `cReadModel` interface variants
- **Table Management**: `cTable`, `cTableData`, `cTableCell`, `cTableMatrix`

- **Formatting**: `cResultTableBuilder` , `cFormatData`
- **Graph Display**: `cGraph*` variants

8. Design Benefits

8.1 Simplified User Interface

Base Functions hide the complexity of the class hierarchy, providing:

- Consistent function signatures
- Standardized parameter validation
- Unified error handling
- Common result presentation patterns

8.2 Maintainability

The Base Function layer provides:

- Single point of API changes
- Consistent parameter handling
- Centralized error message management
- Standard documentation patterns

8.3 Flexibility

Users can choose between:

- **Interactive Tools**: GUI applications for exploration
- **Base Functions**: Simple, consistent interface
- **Object Methods**: Direct class access for advanced users

9. Conclusion

TaesLab's Base Functions represent a sophisticated **facade pattern** that successfully abstracts the complexity of the underlying object-oriented architecture. They serve as the primary user interface while maintaining full access to the powerful class hierarchy underneath.

Key architectural strengths:

1. **Consistent Interface**: All functions follow the same patterns for parameters, validation, and error handling
2. **Robust Error Handling**: Integration with `cMessageLogger` provides detailed error tracking and user feedback
3. **Flexible Usage**: Functions work independently or as part of comprehensive analysis workflows
4. **Extensible Design**: New analysis types can be added following established patterns

The Base Functions successfully balance **ease of use** for novice users with **full functionality** for expert users, making TaesLab accessible across different user skill levels while maintaining the power and flexibility of its sophisticated class architecture.