The dynenv Class and dynenv Package

David Hammen
Odyssey Space Research, LLC

06/06/11

Abstract

All JEOD model documents use the dynenv suite to simplify document specification and to give the document package a uniform appearance. The suite provides tools that aid in document production, in naming and referencing other JEOD models, and in specifying and referencing requirements and IV&V artifacts.

Contents

1 Introduction

This note describes the key capabilities of the JEOD LaTeX dynenv class and dynenv package. All JEOD model documents must use either the dynenv class or package. The suite has four primary purposes:

- Simplify the process of specifying a model document.
- Optionally load a common set of packages. This reduces the work and knowledge burden on model document authors and helps provide a common look to the JEOD documentation suite.
- Provide a standardized way to name and reference JEOD models and JEOD documents.
- Provide a standardized way to represent and reference requirements, inspections, and tests.

2 Using the Suite

The dynenv suite can be used as a class or as a package. The syntax of the two approaches is quite similar:

```
\documentclass[<options>]{dynenv}
```

versus

```
\usepackage[<options>]{dynenv}
```

2.1 The dynenv Class

The dynenv class is a very simple LaTeX class. The body of `dynenv.cls`, sans comments is:

```
\LoadClass[twoside,11pt,titlepage]{report}

\PassOptionsToPackage{complete}{dynenv}
\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{dynenv}}
\ProcessOptions\relax

\RequirePackage{dynenv}
```

The class re-implements the report class with a fixed set of options, currently `twoside,11pt,titlepage`, and then imports the dynenv package. All options to the dynenv class, plus the `complete` option, are passed on to the dynenv package.

2.2 The dynenv Package

The dynenv package is the workhorse of the dynenv suite. The remainder of this document describes features provided by the dynenv package.

2.3 Recommended Practice

The recommended practice is to use the dynenv class in conjunction with the document production macros described in section 5. Doing so results in a very short and very comprehensible main document.

2.4 Class and Package Options

All options provided to the dynenv class are passed on without processing to the dynenv package. The options to the dynenv package fall into two categories: Options that have an explicit option handler, and arbitrary options that do not have a handler. The options with explicit handlers are:

`sigpage` Passed to the dyncover package. When specified, the document will have a signature page.

`nosigpage` Passed to the dyncover package. When specified, the document will not have a signature page. Model documents should not specify either the `sigpage` or `nosigpage` option. They should instead rely on the default.

`section` Indicates that the highest-level sectioning command in the document is `\section`. This document, for example. Model documents should never use the `section` option.

`sections` An alias for `section`.

`part` Indicates that the highest-level sectioning command in the document is `\part`. The documents for large models are sometimes written in parts to make the document more manageable.

`parts` An alias for `part`.

`appendix` Specifying this option causes the dynenv suite to load the appendix package and modifies the behavior of the `backmatter` macro to suit the use of that package.

`complete` Causes the dynenv package to load a set of commonly used packages, the auto-generated `paths.def` file, the model-specific package, and finally, the hyperref package. (The hyperref should always be the very last package that is loaded in any LaTeXdocument.)

`full` An alias for `complete`.

`hyperref` This is explicitly marked as an illegal option. Never specify `hyperref` as an option.

All options other than those listed above indicate additional packages to be loaded by the dynenv package. These additional packages are loaded without any options. If a model document needs to use some non-standard package with options specified, the model-specific package file should use the `\RequirePackage` macro to load the package.

3 Packages and Files Loaded by the dynenv Package

The dynenv package loads a number of standard LaTeX packages plus additional packages and files as indicated by the options to the dynenv class or package. These packages and files are described below, in the order in which they are loaded by the dynenv package. The descriptions below are brief. Users who want to obtain more information on one of the standard packages are encouraged to read the documentation provided by the package developer by typing `texdoc <package_name>` on the command line.

3.1 Package jeodspec

This package identifies the current JEOD release, defines some simple shortcut macros, and specifies the models that comprise the current release. See section 4 for details.

3.2 Package geometry

Specifying page layout in LaTeX is a bit tricky and error-prone. The geometry package solves these issues. The dynenv package uses the geometry package to specify a standard page layout for JEOD documents.

3.3 Package xspace

Consider the shortcut macro `\newcommand{fbz}{foo bar baz}`. Users of this macro have to take care to add a hard space after invoking the macro in the middle of a sentence but never to do so before punctuation. The xspace package solves this problem. Define the macro as `\newcommand{fbz}{foo bar baz\xspace}` and voila, there is no need for that hard space.

3.4 Package xkeyval

The xkeyval package provides a convenient mechanism for specifying and processing options to a class, package, or macro. The dynenv package uses the capabilities provided by xkeyvals to specify and process the options to the document production macros (see section 5), plus a few other macros.

3.5 Package longtable

The longtable package provides the ability to create tables that span multiple pages. Several of the automatically generated tables in JEOD model documents are long tables.

3.6 Package appendix

The appendix package alters some of the base LaTeX mechanisms for presenting appendices and also provides the ability to have end of chapter appendices.

The appendix package is not a part of the standard or extended sets of packages loaded by the dynenv package. The package is loaded only if the `appendix` option is provided to the dynenv class or package. The appendix package should not be used in documents with parts.

3.7 AMS Math Packages amsmath, amssymb, and amsfonts

The amsmath package improves upon and augments the base LaTeX mechanisms for representing and displaying mathematical formulas and is the basis for the American Mathematical Society LaTeX suite. The amssymb package defines miscellaneous mathematical symbols beyond those provided by the amsmath package, and the amsfonts package provides things such as blackboard bold letters, Fraktur letters, and other miscellaneous symbols.

The AMS math packages are not a part of the standard set of packages loaded by the dynenv package but they are a part of the extended set of packages (those enabled by specifying the dynenv `complete` option).

3.8 Package graphicx

The graphicx package provides a range of capabilities. In JEOD, the graphics bundle is used primarily to include graphics into a document via the `includegraphics` macro. The graphicx package is not a part of the standard set of packages loaded by the dynenv package but it is part of the extended set of packages, enabled by specifying the dynenv `complete` option.

3.9 User-Specified Packages

All options provided to the dynenv package other than those explicitly handled by a option handler specify some non-standard package to be loaded. See section 2.4 for a list of options that have explicit handlers.

One side effect of this handling of dynenv package options is that options that do have explicit handlers cannot be used to load a package. That said, only three packages in the LaTeX distribution are shadowed by dynenv package options:

`appendix` The dynenv package loads the appendix package when the `appendix` option is specified, so this shadowing doesn't count.

`hyperref` The dynenv package explicitly makes `hyperref` an illegal option because the hyperref package by design needs to be the very last package loaded in a document. That the hyperref package is shadowed by the `hyperref` option is intentional.

`section` The section package changes the appearance of sectioning commands. JEOD documents should not use this package.

The use of the extended options capability is optional. The mechanism is a bit opaque. Some authors would rather explicitly invoke the `\RequirePackage` macro in the model-specific package file. Another issue is that the packages loaded via this mechanism are loaded without options. If options need to be specified those packages must be loaded in the model-specific package file.

3.10 Package dynmath

The dynmath package provides macros that implement the recommended practice for displaying vectors, matrices, and quaternions. This package is not a part of the LaTeX distribution but is a part of the standard set of packages loaded by dynenv. Documentation for this package is available in the JEOD file `docs/templates/jeod/dynmath.pdf`.

3.11 Package dyncover

The dyncover package creates cover pages for JEOD documents. This package is not a part of the LaTeX distribution but is a part of the standard set of packages loaded by dynenv. Documentation for this package is, ummm, RTFC.

3.12 File paths.def

The standard model document makefile automatically generates the file `paths.def`. This file defines a standard set of macros whose values depend on the model and on the make target. (The relative paths lose one level of indirection when the file is built with `make install` versus `make`.) The names defined in this file are used in `dynenv.sty` in various places.

As a fictitious example, consider the model located in `$JEOD_HOME/models/interactions/psychoceramics`. The contents of the `paths.def` file for this model given the command `make` will be:

```
\ifx\model@pathsdef\endinput\endinput\else\let\model@pathsdef\endinput\fi|\\
\newcommand\JEODHOME{../../../../..}
\newcommand\MODELHOME{../..}
\newcommand\MODELDOCS{..}
```

```
\newcommand\MODELPATH{models/interactions/psychoceramics}
\newcommand\MODELTYPE{interactions}
\newcommand\MODELGROUP{interactions}
\newcommand\MODELDIR{psychoceramics}
\newcommand\MODELNAME{psychoceramics}
\newcommand\MODELTITLE{\PSYCHOCERAMICS}
\endinput
```

The dynenv package loads this file via `\input{paths.def}` when the package is operating in `complete` mode. The burden of loading this file otherwise falls on the model document author. Note that the file can safely be included multiple times because the first line of `paths.def` is the TEX equivalent of the one-time include protection used in C++ header files.

3.13 Model-Specific Package

One such place where the dynenv package uses one of names defined in `paths.def` is the very next operation the dynenv package performs when the package is operating in `complete` mode: It loads the model-specific package file via `\RequirePackage{\MODELNAME}`. The model-specific package file defines a standard set of macro names whose definitions are model-specific, plus additional macros that are specific to the model. See section 5.5.2 for details.

3.14 Package hyperref

The hyperref package makes all cross-references in a PDF document into hyperlinks and adds the ability to insert hyperlinks to other documents. The hyperref package is not a part of the standard set of packages loaded by the dynenv package but it is part of the extended set of packages, enabled by specifying the dynenv `complete` option.

To make the content linkable, the package needs to modify many macros created by other packages. The hyperref is typically the very last package loaded in a LATEX document to ensure that it can make those alterations. The dynenv package loads hyperref when the dynenv package is used per the recommended practice. This means that a `.tex` file that follows the recommended practice must by necessity have a very simple LATEX preface. There should be no `\usepackage` statements in the main `.tex` file for a model document that follows the recommended practice.

4 Package jeodspec

The jeodspec package defines commands that need to be updated with every release and with additions to or deletions from the JEOD model suite. The maintainers of this package will be JEOD administrators and model developers rather than TEX programmers. The contents of `jeodspec.sty` are designed with this in mind. Extensive knowledge of TEX is not required.

The first set of definitions in `jeodspec.sty` specify items that need to be changed with each release:

`\JEODid` The name of the current (or upcoming) JEOD release.

`\RELEASEMONTH` The name of the month when the release will occur.

`\RELEASEYEAR` The four-digit year when the release will occur.

The next set of definitions specify items that need less frequent changes:

\JEODMANAGERS The names and titles of the people who need to approve the JEOD documentation suite.

\JEODORG The JSC organization under which JEOD is developed and maintained.

\JEODJSCNUM The official JSC document number for the JEOD documentation suite.

The third set of definitions create shortcut commands for text phrases used in multiple documents. Some comments:

- Model authors: Please do not add model-specific shortcuts to `jeodspec.sty`. These shortcuts should be for widely-used terminology. The model-specific .sty file is the place to define such shortcuts.
- The expansion of the older items in this set do not end with \xspace. Please consider using \xspace in future definitions.

The final set of definitions in `jeodspec.sty` identify the models that comprise the current JEOD release. Each JEOD model is specified via the \addmodel macro. \addmodel takes four arguments, as follows:

- The model name, by convention an all-caps, letters-only identifier. For example, this argument is `ATMOSPHERE` for the atmosphere model.
- The model parent directory name, one of dynamics, environment, interactions, or utils. The atmosphere model is located in `$JEOD_HOME/models/environment/atmosphere`, so in this case the parent directory name is `environment`.
- The model directory name. For the atmosphere model this is `atmosphere`. Do not escape underscores in the model directory name. The model directory name for the Earth Lighting Model is `earth_lighting`, not `earth\_lighting`.
- The model title. This is often an initial caps version of the model name, plus "Model". For example, the title of the atmosphere model is `Atmosphere Model`.

5 Document Production

The `dynenv` package provides three macros that simplify the specification of a document, `frontmatter`, `mainmatter`, and `backmatter`. These macros write the front matter, main matter, and back matter of a JEOD document.

5.1 The Structure of a JEOD Document

All JEOD document must have:

- Front matter than comprises:
  - The cover pages,
  - An abstract or executive summary,
  - A table of contents, and
  - Optional lists of figures and tables.
- Body matter that comprises five chapters labeled:
  - Introduction,

  - Product Requirements,
  - Product Specification,
  - User Guide, and
  - Inspections, Tests, and Metrics.

  Details regarding the contents of these chapters is specified elsewhere. Documents that have multiple parts should have these five chapters in each part.

- Back matter that comprises:
  - Optional appendices, and
  - A bibliography.

  The bibliography is constructed automatically from the references in the document, the JEOD BibTeX file, and the model-specific BibTeX file.

5.2 Front Matter

The `\frontmatter` macro constructs the front matter. The macro takes one optional and one mandatory argument.

```
\frontmatter[options]{abstractOrSummary}
```

The options are a comma-separated list of elements of the form `key=value` or just `key`. The keys are:

`style=<style>` Specifies whether the document summary is written using the abstract environment or as an unnumbered chapter. Use `style=abstract` for the former, `style=summary` for the latter.

`label=<name>` Specifies the PDF label of the summary. This option is only needed when the summary is written as an unnumbered chapter and the name of that chapter is something other than "Executive Summary."

`figures` Causes the list of figures to be generated.

`tables` Causes the list of tables to be generated.

Note: The optional argument is not completely optional. You must specify the `style`.

The mandatory argument specifies the name of the file (sans the `.tex` suffix) the contains the abstract or summary.

5.2.1 Sample Usage

The following generates front matter with a list of tables (but no list of figures) with the abstract contained in the file `ModelAbstract.tex`.

```
\frontmatter[style=abstract,tables]{ModelAbstract}
```

The next example generates front matter with a list of figures and a list of tables with an executive summary contained in the file `summary.tex`.

```
\frontmatter[style=summary,figures,tables]{summary}
```

The final example generates front matter with no list of figures or list of tables with an abstract in `chapter*` form contained in the file `abstract.tex`.

```
\frontmatter[style=summary,label=Abstract]{abstract}
```

5.2.2 Abstract or Executive Summary?

The purpose of the abstract or executive summary is to inform the reader of the relevance of the subject of the document to some problem the reader needs to solve. This preface to the document should clearly but concisely summarize the subject of the document.

Use an abstract if this summary description is very brief, four or five paragraphs at most. An abstract cannot contain any sectioning commands. Longer descriptions that need some internal structure should be written in an executive summary form.

An executive summary can be considerably longer than four or five paragraphs but should never be more than 10% of the document as a whole. An executive summary must at a minimum start with a `\chapter*` command and may contain `\section*` and lower level sectioning commands. Always use the starred form of the sectioning commands in an executive summary. While the outline of the main matter is highly structured, this is not the case for an executive summary. You don't need no stinkin' requirements, for example. Write in a manner that best summarizes the material.

5.2.3 Sample Abstract

The following sample abstract summarizes the fictitious psychoceramics model.

```
\begin{abstract}
The cracked pot interaction, first investigated by Josiah S. Carberry, is
typically a small perturbative force in spacecraft dynamics. Refinements of
the interaction have been proposed over the years by a number of authors.
Future developments in this field may well revolutionize space flight.
This is the golden age of psychoceramics! The Psychoceramics Model
implements models of several of the more promising of these techniques.
\end{abstract}
```

Assuming the above is the contents of the `abstract.tex` file in the `docs/tex` directory of the psychoceramics model, the toplevel `psychoceramics.tex` file could include this abstract via

```
\frontmatter[style=abstract,tables]{abstract}
```

5.2.4 Sample Executive Summary

The executive summary form of the preface provides a more detailed (but still summary) description of the subject matter. An executive summary for the above fictitious model is presented below.

```
\chapter*{Executive Summary}
\section*{Overview}
The cracked pot interaction, first investigated by Josiah S. Carberry, is
typically a small perturbative force in spacecraft dynamics. Refinements of
the interaction have been proposed over the years by a number of authors.
```

```
Future developments in this field may well revolutionize space flight.
This is the golden age of psychoceramics! The Psychoceramics Model
implements models of several of the more promising of these techniques.

\section*{Implemented Interactions}
A wide variety of interactions are implemented in the model, far too many to
enumerate in this introductory note. One of the more interesting is the
application of Bell's inequality to a vehicle at 37.235163 north latitude,
115.810645 west longitude on the 91st day of the year (92nd in leap years).
For other interactions provided by the model, users are advised to RTFC.

\section*{Implementation Techniques}
A variety of innovative programming techniques were employed to model the
more esoteric aspects of psychoceramics. The model makes extensive use of
obfuscatory techniques, several of which have been featured in the IOCCC.
Multiple virtual protected inheritance, quadruple dispatch, nondeterministic
behavior, and operator re-re-overloading are heavily employed. The model
explores the deepest and darkest corners of metaprogramming, and implements
features that hitherto have been lost since the days of INTERCAL and APL.
```

Assuming the above is the contents of the `summary.tex` file in the `docs/tex` directory of the psychoceramics model, the toplevel `psychoceramics.tex` file could include this executive summary via

```
\frontmatter[style= summary,tables]{summary}
```

5.3 Main Matter

The `\mainmatter` macro constructs the body matter. The macro takes a single argument, a comma-separated list of files to be `\input`. One approach to organizing the document content is to have one master file that encapsulates all of the body content. For example, with the body matter encapsulated in the file `ModelBody.tex`, one would use the following to build the body of the document:

```
\mainmatter{ModelBody}
```

Another approach is to place each of the five chapters in its own `.tex` file. For example, if these five files are `intro.tex`, `reqt.tex`, `spec.tex`, `guide.tex`, and `ivv.tex`, one would use the following to build the body of the document:

```
\mainmatter{intro,reqt,spec,guide,ivv}
```

5.3.1 Introduction

The first chapter of the document introduces the subject of the document in a structured way. This chapter comprises four sections:

1.1 (Purpose and Objectives): Brief description of the subject matter.

1.2 (Context within JEOD): Context of the subject matter within JEOD.

1.3 (Document History): History of the document.

1.4 (Document Organization): Table of contents in text form.

The bulk of this chapter is standard across almost all model documents. Only sections 1.1 (purpose and objectives) and 1.3 (document history) are document-specific.

The macro `\boilerplatechapterone` constructs this first chapter. The macro takes two arguments, the contents of section 1.1 (purpose and objectives) and the table entries for section 1.3 (history). Example:

```
\boilerplatechapterone{
  The cracked pot interaction, first investigated by Josiah S. Carberry, is
  typically a small perturbative force in spacecraft dynamics. Refinements of
  the interaction have been proposed over the years by a number of authors.
  Future developments in this field may well revolutionize space flight.
  This is the golden age of psychoceramics! The Psychoceramics Model
  implements models of several of the more promising of these techniques.}
{
  \ModelHistory
}
```

In the above example, the first argument is the first paragraph from the executive summary. The second paragraph is the `\ModelHistory` command, presumably defined in the model-specific .sty file. See section 5.5.2.

5.3.2 Product Requirements

Unlike the other four chapters, the requirements chapter has no sections. It comprises one requirement after another. Example:

```
\requirement{Flux Capacitor}
\label{reqt:1.21gigawatts}
\begin{description:}
\item[Requirement]
 The model shall simulate the behavior of a vehicle outfitted with a
 flux capacitor connected to a 1.21 gigawatt power source
 as the vehicle crosses the 88 MPH threshold.
\item[Rationale]
This capability is needed by the BTTF simulation group.
\item[Verification]
Inspection, test
\end{description:}
```

5.3.3 Product Specification

The product specification chapter of the document introduces the subject of the document in a structured way. This chapter comprises four sections:

3.1 (Conceptual Design): This section provides a conceptual description of the model. This section typically provides answers to the following:

- What are the key concepts of the model?
- What is the model architecture?
- What drove the design of the model?

3.2 (Mathematical Formulations): This section summarizes the mathematics employed by the model. In the case of a mathematics-free computer science model, this section should be named "Key Algorithms."

3.3 (Interactions): Describes how the model interacts with other JEOD models and other external agents.

3.4 (Detailed Design): Details on the design.

3.5 (Inventory): Details on the design.

5.3.4 User Guide

4.1 (Instructions for Simulation Users): Broad description of classes, key design concepts, etc.

4.2 (Instructions for Simulation Developers): Broad description of classes, key design concepts, etc.

4.3 (Instructions for Model Developers): Broad description of classes, key design concepts, etc.

5.3.5 Inspections, Tests, and Metrics

5.1 (Inspections): Broad description of classes, key design concepts, etc.

5.2 (Tests): Broad description of classes, key design concepts, etc.

5.3 (Requirements Traceability): Broad description of classes, key design concepts, etc.

5.4 (Metrics): Broad description of classes, key design concepts, etc.

5.3.6 Multi-part Documents

Word.

5.4 Back Matter

The `\backmatter` macro constructs the back matter. The macro takes one optional and one mandatory argument.

```
\backmatter[options]{appendices}
```

The options are a comma-separated list of elements of the form `key=value` or just `key`. The keys are:

`bibpos=<pos>` Specifies whether the bibliography is placed before (`pos=start`) or after (`pos=end`) the appendices.

`modelbibfile=<name>` Specifies the name of the model-specific `.bib` file. This option is needed only in the case that the model author has not followed the recommended practice of naming the model-specific `.bib` file after the model.

`nomodelbibfile` Specifies that there is no model-specific `.bib` file.

The mandatory argument specifies the names of the files (sans the `.tex` suffix) the build the appendices. Use an empty argument '' if the model has no appendices.

5.5 Document Directory Files

5.5.1 Makefile

5.5.2 Model-Specific Package

when the dynenv package is operating in complete mode, it loads the model-specific package file via \RequirePackage{\MODELNAME}

One such place where the dynenv package uses one of names defined in paths.def is the very next operation the dynenv package performs when the package is operating in complete mode: It loads the model-specific package file via \RequirePackage{\MODELNAME}. For a model document that is built following the recommended practice, the model-specific package file must define the following:

\ModelRevision The revision number of the model document.
: This should be incremented in some manner when the contents of the model document are updated. It is recommended that this definition be placed very close to the start of the model-specific package to ease the update process.

\ModelAuthor The name or names of the people who wrote the current revision of the model.
: The \ModelAuthor is used to generate the inside cover and is stored as the PDF author of the document. Multiple authors, if present, must be separated by newlines (\\).

\ModelPrefix A one word name of the model.
: The \ModelPrefix is used to label the requirements, inspections, and tests that are specified in the model's .tex files.

\ModelDesc A short description of the model.
: The \ModelDesc is stored as the PDF description of the document.

\ModelKeywords A comma-separated list of keywords.
: The \ModelKeywords are stored as the PDF keywords of the document.

It is recommended that the document history also be stored in the model-specific package file as the command \ModelHistory. This practice places the document revision number and history in close proximity, thereby easing update. The command should define a list of table entries, newest to oldest. Each entry must be of the form Author & Date & Revision & Description \\. The fields are:

Author The authors responsible for the current document revision.

Date The month and year of the document revision.

Revision The document revision number in major.minor format.

Description A brief description of why the revision was made.

A sample model-specific style file follows for the fictitious psychoceramics model.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Model-specific style sheet for the Psychoceramics Model.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\NeedsTeXFormat{LaTeX2e}[2009/01/01]% LaTeX date must be January 2009 or later

% Package definition.
```

```
\ProvidesPackage{psychoceramics}[2011/03/01 v1.0 psychoceramics model macros]

% Author and revision number of current revision of the document.
\newcommand\ModelAuthor{Josiah S. Carberry, III}
\newcommand\ModelRevision{1.0}

% Document history, in the form of a list of table entries.
% Entries should be in reverse time order (newest first).
% Each entry must be of the form
%   Author & Date & Revision & Description \\
\newcommand\ModelHistory {
  Josiah S. Carberry, III & April 2011 & 1.0 & Initial version \\
}

% Model description macros, used for the cover page and to tag requirements.
\newcommand\ModelPrefix{Psychoceramics}
\newcommand\ModelDesc{Psychoceramics Model\xspace}
\newcommand\ModelKeywords{BEM, FTL, BTTF}
```

5.5.3 Top-level .tex File

Exhibit 5.1 displayes the complete contents of the top-level .tex file file the fictitious psychoceramics model.

5.5.4 Lower Level .tex File

6 Requirements, Inspections, and Tests

One of the key strengths of the JEOD project is the rigor of its verification and validation process. This section describes the macros that authors must use in their model document `.tex` files to specify the requirements of the model, the inspections of the model against the requirements, and the software tests that demonstrate that the model satisfies the requirements levied against it.

6.1 Requirements

The `\requirement` macro is used in the Requirements chapter of a model document to introduce a new requirement. The macro takes one argument, the name of the requirement. After introducing a requirement, give it a label and then describe the requirement in a description block with a very specific form. For example,

```
\requirement{Handling of Incoming Events}
\label{reqt:unique_label}
\begin{description}
\item[Requirement:]\ \\
  The model shall do this, that, and the other upon receipt of an event.
\item[Rationale:]\ \\
  This is cool, that is needed, and the other is just awesome.
\item[Verification:]\ \\
  Inspection, test
\end{description}
```

Exhibit 5.1: Contents of `psychoceramics.tex`

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% psychoceramics.tex
% Top level document for the Psychoceramics Model
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Use the standard JEOD dynenv document class
\documentclass{dynenv}

% Add the requirements et al. to the table of contents
\dynenvaddtocitems{reqt=1,insp=2,test=2}

\begin{document}

% Front matter: Executive summary, table of contents, lists of figures & tables
\frontmatter[style=summary,figures,tables]{summary}

% Main document body: Each chapter is in its own file
\mainmatter{intro,reqt,spec,guide,ivv}

% Back matter: Bibliography
\backmatter{}

\end{document}
```

The above generates the following:

Requirement Example_1: Handling of Incoming Events

Requirement:
The model shall do this, that, and the other upon receipt of an event.

Rationale:
This is cool, that is needed, and the other is just awesome.

Verification:
Inspection, test

The requirement can be cross-referenced using the `\label` given to the requirement. For example, `requirement~\ref{reqt:unique_label}` yields requirement Example_1. The requirements can be inserted in the table of contents if desired (see section 6.6).

One problem with the way in which requirement Example_1 was written is that the use of the description environment forces the document author to write description item labels in a peculiar way. The author needs to ensure item labels end with a colon and needs to add that `\ \\` construct after the label to force a newline. The `dynenv` package provides a colon form of the description environment, `description:`, to automatically perform both of these tasks:

```
\requirement{Handling of Incoming Events}
\label{reqt:another_label}
\begin{description:}
\item[Requirement]
  The model shall do this, that, and the other upon receipt of an event.
\item[Rationale]
  This is cool, that is needed, and the other is just awesome.
\item[Verification]
  Inspection, test
\end{description:}
```

Except for the auto-incremented requirement number, the output generated by the above is identical to that of requirement Example_1:

Requirement Example_2: Handling of Incoming Events

Requirement:
The model shall do this, that, and the other upon receipt of an event.

Rationale:
This is cool, that is needed, and the other is just awesome.

Verification:
Inspection, test

There is a bigger problem with this requirement. It is a good example of a bad requirement. The problem is that the requirement specifies three distinct things. One alternative is to split this poorly written requirement into three separate requirements. Another is to use the `\subrequirement` macro:

```
\requirement[Event Handling]{Handling of Incoming Events}
\label{reqt:yet_another_label}
\begin{description:}
\item[Requirement]
 Upon receipt of an event,
 \subrequirement{This.}\label{reqt:yet_another_label_this}
 The model shall do this.
 \subrequirement{That.}\label{reqt:yet_another_label_that}
 The model shall do that.
 \subrequirement{Other.}\label{reqt:yet_another_label_other}
 The model shall do the other.
\item[Rationale]
  This is cool, that is needed, and the other is just awesome.
\item[Verification]
  Inspection, test
\end{description:}
```

Requirement Example_3: Handling of Incoming Events

Requirement:
Upon receipt of an event,

3.1 This. The model shall do this.

3.2 That. The model shall do that.

3.3 Other. The model shall do the other.

Rationale:
This is cool, that is needed, and the other is just awesome.

Verification:
Inspection, test

6.2 Inspections

The `\inspection` macro is used in the Inspections section of the Inspections, Tests, and Metrics chapter of a model document to introduce a new inspection. An "inspection" of a product somehow verifies without using the product that the product satisfies some of the requirements levied on it. Examples include peer reviews, a desk check comparison of the mathematics as implemented in the code against the mathematical description in the model document, and a desk check comparison of the model interfaces versus some external specification.

The `\inspection` macro takes one argument, the name of the inspection. After introducing an inspection, give it a label via the `\label` macro and then describe the inspection in some way. This description can vary in form. The JEOD project has not prescribed a rigid form for these inspections. The inspection must indicate the application requirements and must indicate whether the model passed or failed the inspection.

```
\inspection{Event Processing Inspection}
\label{inspect:events}
The model processes events via the method \verb#EventHandler::process_event#.
This function in turn calls three methods to process the incoming event:
\begin{itemize}
```

```
\item \verb#EventHandler::do_this#, which does this.
\item \verb#EventHandler::do_that#, which does that.
\item \verb#EventHandler::do_the_other_thing#, which does the other thing.
\end{itemize}
By inspection, the model satisfies
requirement~\traceref{reqt:yet_another_label}.
```

The above generates the following:

Inspection Example_1: Event Processing Inspection

The model processes events via the method `EventHandler::process_event`. This function in turn calls three methods to process the incoming event:

- `EventHandler::do_this`, which does this.
- `EventHandler::do_that`, which does that.
- `EventHandler::do_the_other_thing`, which does the other thing.

By inspection, the model satisfies requirement Example_3.

6.3 Tests

The `\test` macro is used in the Tests section of the Inspections, Tests, and Metrics chapter of a model document to introduce a new inspection. A "test" of a product uses the product in some way to demonstrate that the product satisfies some of the requirements levied on it. Some tests involve simulations while others involve unit tests. The test should exercise the model in a manner that allows the output to be compared against some expected result.

The `\test` macro takes one argument, the name of the test. inspection. After introducing a test, give it a label and then describe the test in some way. As with inspections, this description can vary. A widely used approach is to use a description environment. Typical labels are:

Background Provides background information regarding the test.

Test description Provides details on what was done.

Test directory Specifies where the test code resides.

Success criteria Specifies how the test is deemed to be successful.

Test results Describes the results of running the test.

Applicable requirements Identifies the requirement(s) tested.

6.4 Specifying Applicable Requirements

Words

6.5 Traceability

Each inspection and test should trace to one or more requirements, and each requirement should be verified/validated by one or more inspection or test. Every model document is supposed to contain a requirements traceability table[1] that lists the mapping from requirements to inspections and tests.

6.6 Adding Requirements, Inspections, and Tests to the Table of Contents

The `\dynenvaddtotoc` and `\dynenvaddtocitems` macros provide the ability to make the requirements, inspections, and tests specified in document listed in that document's table of contents.

Words

7 Commands and Environments

7.1 Commands

`\addmodel {<NAME>}{<dir>}{<name>}{<title>}` [dynenvupfront.sty]
Word

`\backmatter [<options>]{<list,of,files>}` [dynenvmatter.sty]

`\boilerplatechapterone{<description>}{<history>}` [dynenvboilerplate.sty]
Argument #1 - Contents of section 1.1
Argument #2 - Table entries for history table.
See section 5.3.1 for details.

`\boilerplateinventory` [dynenvboilerplate.sty]
Word

`\boilerplatemetrics` [dynenvboilerplate.sty]
Word

`\boilerplatetraceability` [dynenvboilerplate.sty]
Word

`\escapeus {<text_with_underscores>}` [dynenvupfront.sty]
Word

`\iflabeldefined {<label>}{<commands>}` [dynenvboilerplate.sty]
Argument #1 - Label to be tested.
Argument #2 - Code to be expanded.
Expands second argument if label specified by first argument is defined.

`\JEODHOME` [paths.def]
Path to $JEODHOME.

`\frontmatter [<options>]{<abstractOrSummary>}` [dynenvmatter.sty]

[1] The requirements traceability table is typically located in the "Requirements Traceability" section of the "Inspections, Tests, and Metrics" chapter of a model document.

`\inspection {<name>}` [dynenvreqt.sty]
Words.

`\longentry` [dynenvboilerplate.sty]
Causes broken lines in a `longtable` environment to be printed with a hanging indent.

`\mainmatter [<options>]{<list,of,files>}` [dynenvmatter.sty]

`\MODELDIR` [paths.def]
Model directory, with underscores escaped.

`\MODELDIRx` [dynenv.sty]
`{\MODELDIR\xspace}`

`\MODELDOCS` [paths.def]
Relative path to model documentation directory.

`\MODELGROUP` [paths.def]
Model group (same as type except for utils).

`\MODELGROUPx` [dynenv.sty]
`{\MODELGROUPx\xspace}`

`\ModelHistory` [<model_name>.sty]
History table entries.

`\MODELHOME` [paths.def]
Relative path to model directory.

`\MODELNAME` [paths.def]
Model directory, underscores not escaped.

`\MODELPATH` [paths.def]
Path to model directory from $JEODHOME.

`\MODELPATHx` [dynenv.sty]
`{\MODELPATH\xspace}`

`\ModelPrefix` [<model_name>.sty]
One-word prefix for requirements, etc.

`\MODELTITLE` [paths.def]
All-caps model name command.

`\MODELTITLEx` [dynenv.sty]
`{\MODELTITLE\xspace}`

`\MODELTYPE` [paths.def]
Model type (e.g., dynamics, environment).

`\MODELTYPEx` [dynenv.sty]
`{\MODELTYPE\xspace}`

`\simpletracetable {<one>}{<two>}{<three>}` [dynenvreqt.sty]
Words.

`\requirement {<name>}` [dynenvreqt.sty]
Words.

`\subrequirement {<name>}` [dynenvreqt.sty]
Words.

`\test {<name>}` [dynenvreqt.sty]
Words.

`\traceref {<label>}` [dynenvreqt.sty]
Words.

`\tracerefrange {<label1>}{<label2>}` [dynenvreqt.sty]
Words.

`\tracetable {<one>}{<two>}{<three>}` [dynenvreqt.sty]
Words.

7.2 Environments

`abstract` [dynenvmatter.sty]
The abstract environment defined in `report.cls` issues an almost harmless `\titlepage` command, the harm being that it resets the page number to one (or rather, roman i). Redefining this environment avoids this problem.

`codeblock` [dynenvcode.sty]
The codeblock environment is a specialization of the Verbatim environment. Use this environment for code to be printed inline with the text. The printed code will be indented with respect to the current indentation level.

`codebox` [dynenvcode.sty]
The codebox environment is a specialization of the Verbatim environment. Use this environment for code to be printed in an exhibit. The code will be framed in a box and is indented slightly with respect to page boundaries.

`description:` [dynenvreqt.sty]
The description: environment is a specialization of the description environment intended for use with requirements.

`exhibit` [dynenvcode.sty]
The exhibit environment is similar to the table environment. Exhibits, like tables, can have captions and labels. The caption should be placed above the exhibited items. The exhibit will be listed as a part of the list of tables.

`stretchlongtable` [dynenvboilerplate.sty]
A `stretchlongtable` is a `longtable` environment that takes an optional `arraystretch` (default: 1.2). If specified, the `arraystretch` value must be specified in angle brackets and must precede the optional arguments to the `longtable` environment.