SPM5 to Matlabbatch Transition Guide

March 4, 2008

1 Overview

This is a short overview to describe code organisation and interfaces between SPM and the batch system.

1.1 Code Reorganisation

The following paths have changed:

- `fullfile(spm('dir'),'matlabbatch')` Core batch system.
- `fullfile(spm('dir'),'config')` New SPM config files.
- `fullfile(spm('dir'),'oldconfig')` Old SPM config files (unused)
- `spm_jobman.m` and `spm_select.m` replaced with compatibility code
- `spm_Menu.fig` Callbacks adapted

Configuration code has been generated automatically from the existing SPM configuration using `cfg_struct2cfg` and `gencode`. This sometimes results in redundant/duplicate code. Also, conditional constructs like `if`, `case` may not have been considered.

For details, see section 2.

1.2 Interfaces between SPM and Matlabbatch

Unchanged harvested job structure.

Changed Top-level node in SPM config now called `spmjobs` instead of `jobs`. New overall top-level node `matlabbatch`. `spm_jobman` will convert and load SPM5 style batch jobs into the new batch system.

Changed Configuration file syntax - instead of structs, configuration items are now objects. Structs of type `<type>` are now represented as objects of class `cfg_<type>`. Existing SPM5 configuration can be imported using `cfg_struct2cfg`. There is a new class `cfg_exbranch` which is used for branches that have a `.prog` field.

Deprecated Virtual files have been replaced by dependencies. These require computations to return a single output argument (e.g. a cell, struct). Parts of this output argument can be passed on to new inputs at run-time. Virtual files are treated as a special output argument.

Added Interface to the batch system

- `cfg_util` Configuration management, job management, job execution
- `cfg_serial` A utility to fill missing inputs and run a job (optionally with a GUI input function)
- `cfg_ui` GUI - inspired by `spm_jobman`, but modified to work around some MATLAB GUI "features" (like input widgets loosing focus before editing has finished).

2 Configuration Code Details

Configuration code has been split into two files per configuration:

spm_cfg_*.m Configuration classes, `.check`, `.vout` subfunctions

spm_run_*.m Run-time code, takes job structure as input and returns output structure as specified in `.vout`.

In a few cases (where there was no additional magic in the code), run-time code has been integrated into the main SPM code. This may be useful to run test batches without using the configuration/batch system.

2.1 Virtual Outputs

Virtual outputs are described by arrays of `cfg_dep` objects. These objects contain a "source" and a "target" part. Functions may have more than one virtual output (e.g. one output per session, a collection of results variables). One `cfg_dep` object has to be created for each output.

Only two fields in the "source" part need to be set in a `.vout` callback:

sname A display name for this output. This will appear in the dependencies list and should describe the contents of this dependency.

src_output A subscript reference that can be used to address this output in the variable returned at run-time.

tgt_spec (optional) A description on what kind of inputs this output should be displayed as dependency. This is not very convenient yet, the `match` and `cfg_findspec` methods are very restrictive in the kind of expressions that are allowed.

The `.vout` callback will be evaluated once the configuration system thinks that enough information about the *structure* of the outputs is available. This condition is met, once all in-tree nodes `cfg_(ex)branch`, `cfg_choice`, `cfg_repeat` have the required number of child nodes.

The `.vout` callback is called with a job structure as input, but its code *should not rely* on the evaluation of any contents of this structure (or at least provide a fallback). The contents of the leaf nodes may not be set or may contain a dependency object instead of a value during evalution of `.vout`.

The "target" part will be filled by the configuration classes, the `src_exbranch` field is set in `cfg_util`.

2.2 SPM Startup

The top level configuration file for SPM is `spm_cfg.m`. It collects SPM core configuration files and does toolbox autodetection. If a toolbox directory contains `*_cfg_*.m` files, they will be loaded. Otherwise, if there are only SPM5-style `*_config_*.m` files, the configuration will be converted at run-time using `cfg_struct2cfg`.

2.3 Defaults Settings

Due to automatic code generation from the previous configuration, some defaults are coded in the configuration files. This code will be removed, all defaults will be read from `spm_def.m` in the configuration folder. This file currently links to `spm_defaults.m` and contains code to include toolbox-specific defaults in a way similar to toolbox autodetection (i.e. they appear under `defaults.tools.<toolbox_tag>` in the defaults structure.

2.4 Toolbox Migration

In the `fullfile(spm('dir'),'toolbox')` folder there exists a migration utility `spm_tbx_config2cfg.m`. This utility will create a `*_cfg_*.m` and a `*_def_*.m` file based on the configuration tree given as input argument.

3 Utilities

3.1 Batch Utilities

Matlabbatch is designed to support multiple applications. A standard application "BasicIO" is enabled by default. Among other options, it contains file/file selection manipulation utilities which can be used as as dependency source if multiple functions require the same set of files as input argument. For debugging purposes, "Pass Output to Workspace" can be used to assign outputs of a computation to a workspace variable.

The `cfg_confgui` folder contains an application which describes all configuration items in terms of configuration items. It is not enabled by default, but can be added to the batch system using `cfg_util('addapp'...)`. This utility can be used generate a batch configuration file with the batch system itself.

3.2 Matlab Code Generation

The `gencode` utility generates MATLAB `.m` file code for any kind of MATLAB variable. This is used to save batch files as well as to generate configuration code.

3.3 Configuration Management

The backend utility to manage the configuration data is `cfg_util`. It provides callbacks to add application configurations, and to load, modify, save or run jobs. These callbacks are used by two frontends: `cfg_ui` is a MATLAB GUI, while `cfg_serial` can be used both as a GUI and in script mode. In script mode, it

will fill in job inputs from an argument list. This allows to run predefined jobs with e.g. subject dependent inputs without knowing the exact details of the job structure.