Contents

Never compile this.

```
;; -*- no-byte-compile: t; -*-
```

1. Org-Mode Exemple Complet Minimal

   ID: org_gcr_2017-05-12_mara:1035FF79-3703-49A6-8522-618B38A48F6C

   Configure EMACS to easily provide ECMs.
   Sysop is likely to use this often.
   Start EMACS with this command:

   ```
   emacs --debug-init --no-init-file --no-splash --background-color white --fo
   ```

   (a) Principle of Least Astonishment (POLA)

   ID: org_gcr_2017-05-12_mara:626B5DD1-97D8-4B85-96BC-B9A96F18AF1E

   i. Time

   ID: org_gcr_2018-03-16_mara:BC16A47E-FC4E-4F12-8813-583BF4C3EF5A
   Standardize timestamps.

```
(defun help/get-timestamp ()
  "Produces a full ISO 8601 format timestamp."
  (interactive)
  (let* ((timestamp-without-timezone (format-time-string
   "%Y-%m-%dT%T"))
         (timezone-name-in-numeric-form (
   format-time-string "%z"))
         (timezone-utf-offset
          (concat (substring
   timezone-name-in-numeric-form 0 3)
                  ":"
                  (substring
   timezone-name-in-numeric-form 3 5)))
         (timestamp (concat timestamp-without-timezone
                            timezone-utf-offset)))
    timestamp))
(defun help/insert-timestamp ()
  "Inserts a full ISO 8601 format timestamp."
  (interactive)
  (insert (help/get-timestamp)))
```

```
(defun help/get-timestamp-no-colons ()
  "Produces a full ISO 8601 format timestamp with colons
   replaced by hyphens."
  (interactive)
  (let* ((timestamp (help/get-timestamp))
         (timestamp-no-colons (replace-regexp-in-string "
   :" "-" timestamp)))
    timestamp-no-colons))
(defun help/insert-timestamp-no-colons ()
  "Inserts a full ISO 8601 format timestamp with colons
   replaced by hyphens."
  (interactive)
  (insert (help/get-timestamp-no-colons)))
(defun help/insert-datestamp ()
  "Produces and inserts a partial ISO 8601 format
   timestamp."
  (interactive)
  (insert (format-time-string "%F")))
```

ii. Garbage Collection

`ID: org_gcr_2017-07-29_mara:5A6162AE-F0FD-491D-BC05-F288F46F6125`

Clear memory and disable garbage collection or return garbage collection to normal.

```
(setq help/default-gc-cons-threshold gc-cons-threshold)
(defun help/set-gc-cons-threshold (&optional multiplier
   notify)
  "Set `gc-cons-threshold' either to its default value or
    a
   `multiplier' thereof."
  (let* ((new-multiplier (or multiplier 1))
         (new-threshold (* help/default-gc-cons-threshold
                           new-multiplier)))
    (setq gc-cons-threshold new-threshold)
    (when notify (message "Setting `gc-cons-threshold' to
     %s" new-threshold))))
```

iii. Load Behavior

`ID: org_gcr_2017-05-12_mara:75985F03-F3B9-4DA3-8F6E-393E4C2F06E7`

EMACS can load 3 different representations of a Emacs-Lisp source file code OOTB. The name of source code file is the value before the file extension. When you pass `load` a name it searches for an acceptable representation. Representation types are indicated by their extension name. `.el` is a human readable and uncompiled. `.elc` is not human readable and compiled. `.gz` is Gzip

compressed and contains `.el` or `.elc` files.
The variable `load-suffixes` determines the order for which text and byte-code representations are first searched by extension-name. The variable `load-file-rep-suffixes` determines the order for all other extension types.
OOTB, EMACS combines the productivity of REPL style of development with the speed of compiled code by `load`'ing byte-code first, text second, and compressed third. This workflow gives you the fastest code at run-time and lets you experiment with new features stored in text. When you are ready to use them every time, you compile them. There is only one downside of this approach: when you forget that it works this way it can be confusing.
When you forget about that style of system you end up with surprising behavior. The surprise comes when you forget to compile code and then write new code that depends on the now old version of that code. After the next build, your system can break in surprising ways. This violates the Principle of Least Astonishment.
This system values predictability so it does the simplest thing possible: `load` searches for the newest representation of a file and uses that one. It assumes that Sysop has total and complete control over the management of file representations.
This is the **first** thing that *ought* to happen in the initialization of **any** tangled system.

```
(setq load-prefer-newer t)
```

(b) Org-Mode Exemple Complet Minimal

`ID: org_gcr_2017-05-12_mara:572E2309-5DCA-4AE1-AAC4-36B7E07AD46D`

Every new EMACS releases comes with the latest stable Org-Mode release. To get hot-fixes, cutting edge features, and easy patch creation though, you need to use the version from Git.

These detailed and clear directions explain how ot run Org-Mode from Git. The only thing worth mentioning again is that in order to use **any** version of Org-Mode other than the one that comes OOTB you **must** load Org-Mode **before** anything else in your initialization file. It is easy to do! When you get unexpected Org-Mode behavior be sure to stop and investigate `org-version` and decide whether or not it is what you expect and prepare an ECM if necessary.

i. Things That Must Occur Before Loading Org-Mode

`ID: org_gcr_2017-07-30_mara:3CF35008-D435-4CCB-90D7-5CFA06E15467`

Add the Org-Mode core distribution the load path.

```
(add-to-list 'load-path "~/src/org-mode/lisp")
```

Add the Org-Mode-Contributions distribution to the load path. The contributions are essential.

```
(add-to-list 'load-path "~/src/org-mode/contrib/lisp")
```

Allow single-character alphabetical bullet lists. This configuration must occur before loading Org-Mode. **Never** remove this from a submitted ECM.

```
(setq org-list-allow-alphabetical t)
```

Unchecked boxes prevent marking the parent as done. This configuration must occur before loading Org-Mode. **Never** remove this from a submitted ECM.

```
(setq org-enforce-todo-checkbox-dependencies t)
```

Use math double brackets for Literate Programming instead of GUILLEMET delimeters.

```
(setq org-babel-noweb-wrap-start "⟪")
(setq org-babel-noweb-wrap-end "⟫")
```

ii. Loading Org-Mode

ID: org_gcr_2017-07-30_mara:FFA7E062-C039-4F3F-82FC-12A49FF379B8

Load Org-Mode.

```
(require 'org)
```

iii. Things That Must Occur Only After Loading Org-Mode

ID: org_gcr_2017-07-30_mara:D9207828-3783-4599-BA48-A6EB2C3FCAE4

Helper doubles available memory.

```
(defun help/double-gc-cons-threshold () "Double `
   gc-cons-threshold'." (help/set-gc-cons-threshold 2))
```

Double garbage collection during tangling. Instead of storing this in the primary Org-Mode Literate Programming configuration I want it here so that it is always available.

```
(add-hook 'org-babel-pre-tangle-hook #'help/
   double-gc-cons-threshold)
(add-hook 'org-babel-post-tangle-hook #'help/
   set-gc-cons-threshold)
```

Display system info.

```
(defun help/display-system-info ()
  (interactive)
  (message "<<<ECM Information>>>\nThis buffer file: %s\
   nAs Of: %s\nOrg-Version: %s\nOrg-Git-Version:%s\
   nEmacs-Version: %s\nNoweb wrap start and stop
   delimeters: '%s' and '%s'\
   norg-babel-default-header-args:\n"
           buffer-file-name
           (help/get-timestamp)
           (org-version)
           (org-git-version)
           (emacs-version)
           org-babel-noweb-wrap-start
           org-babel-noweb-wrap-end)
  (pp org-babel-default-header-args))
(help/display-system-info)
```