

```

*****
75913 Sun Nov 16 22:22:25 2008
new/src/updatesmanager.py
*** NO COMMENTS ***
*****
1 #!/usr/bin/python2.4
2 #
3 # CDDL HEADER START
4 #
5 # The contents of this file are subject to the terms of the
6 # Common Development and Distribution License (the "License").
7 # You may not use this file except in compliance with the License.
8 #
9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 # or http://www.opensolaris.org/os/licensing.
11 # See the License for the specific language governing permissions
12 # and limitations under the License.
13 #
14 # When distributing Covered Code, include this CDDL HEADER in each
15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 # If applicable, add the following below this CDDL HEADER, with the
17 # fields enclosed by brackets "[]" replaced with your own identifying
18 # information: Portions Copyright [yyyy] [name of copyright owner]
19 #
20 # CDDL HEADER END
21 #
22 # Copyright 2008 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #

27 import getopt
28 import os
29 import sys
30 import time
31 import locale
32 import gettext
33 import pango
34 import errno
35 from threading import Thread
36 from threading import Timer

38 try:
39     import gobject
40     gobject.threads_init()
41     import gtk
42     import gtk.glade
43     import pygtk
44     pygtk.require("2.0")
45 except ImportError:
46     sys.exit(1)

48 import pkg.client.image as image
49 import pkg.client.api as api
50 import pkg.client.api_errors as api_errors
51 import pkg.client.progress as progress
52 import pkg.gui.beadmin as beadm
53 from pkg.client import global_settings

55 # Put _() in the global namespace
56 import __builtin__
57 __builtin__.__ = gettext.gettext

59 IMAGE_DIRECTORY_DEFAULT = "/" # Image default directory
60 IMAGE_DIR_COMMAND = "svcprop -p update/image_dir svc:/application/pkg/update"
61 CLIENT_API_VERSION = 4 # API version

```

```

62 PKG_CLIENT_NAME = "updatesmanager" # API client name
63 SELECTION_CHANGE_LIMIT = 0.5 # Time limit in seconds to cancel selection upda
64 IND_DELAY = 0.05 # Time delay for printing index progress
65 UPDATES_FETCH_DELAY = 200 # Time to wait before fetching updates, allows g
66 # loop time to start and display main UI
67 #UM Row Model
68 (
69 UM_ID,
70 UM_INSTALL_MARK,
71 UM_STATUS,
72 UM_NAME,
73 UM_REBOOT,
74 UM_LATEST_VER,
75 UM_SIZE,
76 UM_FMRI,
77 ) = range(8)

79 #UPDATE STEPS
80 (
81 UPDATE_EVAL,
82 UPDATE_DOWNLOAD,
83 UPDATE_INSTALL,
84 UPDATE_INDEX,
85 ) = range(4)

87 #UPDATE TYPES
88 (
89 UPDATE_ACTIVE,
90 UPDATE_INACTIVE,
91 UPDATE_DONE,
92 ) = range(3)

94 class GUIProgressTracker(progress.ProgressTracker):
95     """ This progress tracker is designed for Gnome GUI's
96     The parent must provide a number of callback methods to render progress
97     in the GUI context. """

99     def __init__(self, parent):
100         progress.ProgressTracker.__init__(self)
101         self.parent = parent
102         self.__ = gettext.gettext
103
104         self.act_started = False
105         self.ind_started = False
106         self.last_print_time = 0
107         self.dl_started = False
108         self.dl_cur_pkg = None

110     def reset(self):
111         progress.ProgressTracker.reset(self)
112         self.act_started = False
113         self.ind_started = False
114         self.last_print_time = 0
115         self.dl_started = False

116     def cat_output_start(self):
117         catstr = self.__("Fetching catalog: '%s' ..." % (self.cat_cur_cat
118         gobject.idle_add(self.parent.output, "%s" % catstr)

121     def cat_output_done(self):
122         gobject.idle_add(self.parent.output_done, self.__("Fetching catal

124     def eval_output_start(self):
125         s = self.__("Creating Plan ... ")
126         gobject.idle_add(self.parent.output, "%s" % s)

```

```

128 def eval_output_progress(self):
129     if (time.time() - self.last_print_time) >= 0.10:
130         self.last_print_time = time.time()
131     else:
132         return
133     gobject.idle_add(self.parent.progress_pulse)

135 def eval_output_done(self):
136     gobject.idle_add(self.parent.output_done, self._("Creating Plan")
137     self.last_print_time = 0

139 def ver_output(self):
140     if self.ver_cur_fmri != None:
141         if (time.time() - self.last_print_time) >= 0.10:
142             self.last_print_time = time.time()
143         else:
144             return
145         gobject.idle_add(self.parent.progress_pulse)
146         gobject.idle_add(self.parent.output, \
147             self._("Verifying: %s ..") % \
148             self.ver_cur_fmri.get_pkg_stem())
149     else:
150         gobject.idle_add(self.parent.output, "")
151         self.last_print_time = 0

153 def ver_output_error(self, actname, errors):
154     gobject.idle_add(self.parent.output_done, self._("Verifying"))

156 def dl_output(self):
157     gobject.idle_add(self.parent.dl_progress,
158         self.dl_started, self.dl_cur_pkg, \
159         self.dl_cur_npkgs, self.dl_goal_npkgs, \
160         self.dl_cur_nfiles, self.dl_goal_nfiles, \
161         self.dl_cur_nbytes / 1024.0 / 1024.0, \
162         self.dl_goal_nbytes / 1024.0 / 1024.0)

164     if not self.dl_started:
165         self.dl_started = True

167 def dl_output_done(self):
168     self.dl_cur_pkg = self._("Completed")
169     self.dl_output()
170     gobject.idle_add(self.parent.output_done, self._("Download"))

172 def act_output(self):
173     if (time.time() - self.last_print_time) >= 0.05:
174         self.last_print_time = time.time()
175     else:
176         return
177
178     gobject.idle_add(self.parent.act_progress, self.act_started, \
179         self.act_phase, self.act_cur_nactions, self.act_goal_nac

181     if not self.act_started:
182         self.act_started = True

184 def act_output_done(self):
185     self.act_output()
186     gobject.idle_add(self.parent.output_done, self._("Install"))

188 def ind_output(self):
189     if (time.time() - self.last_print_time) >= IND_DELAY:
190         self.last_print_time = time.time()
191     else:
192         return

```

```

194     gobject.idle_add(self.parent.ind_progress, self.ind_started, \
195         self.ind_phase, self.ind_cur_nitems, self.ind_goal_nitem

197     if not self.ind_started:
198         self.ind_started = True
199
200 def ind_output_done(self):
201     self.act_output()
202     gobject.idle_add(self.parent.output_done, self._("Index"))

205 class Updatesmanager:
206     def __init__(self):
207         global_settings.client_name = PKG_CLIENT_NAME
208
209     try:
210         self.application_dir = os.environ["UPDATE_MANAGER_ROOT"]
211     except KeyError:
212         self.application_dir = "/"
213     locale.setlocale(locale.LC_ALL, '')
214     for module in (gettext, gtk.gladefile):
215         module.bindtextdomain("pkg", self.application_dir +
216             "/usr/share/locale")
217         module.textdomain("pkg")
218     # XXX Remove and use _() where self._ and self.parent._ are bein
219     self._ = gettext.gettext

220
221     # Duplicate ListStore setup in get_updates_to_list()
222     self.um_list = \
223         gtk.ListStore(
224             gobject.TYPE_INT,           # UM_ID
225             gobject.TYPE_BOOLEAN,      # UM_INSTALL_MARK
226             gtk.gdk.Pixbuf,            # UM_STATUS
227             gobject.TYPE_STRING,       # UM_NAME
228             gtk.gdk.Pixbuf,            # UM_REBOOT
229             gobject.TYPE_STRING,       # UM_LATEST_VER
230             gobject.TYPE_STRING,       # UM_SIZE
231             gobject.TYPE_STRING,       # UM_FMRI
232         )
233
234     self.progress_stop_thread = False
235     self.initial_active = 0
236     self.initial_default = 0
237     self.last_select_time = 0
238     self.size_thread_running = False
239     self.cancelled = False
240     self.fmri_description = None
241     self.install = False
242     self.install_error = False
243     self.done_icon = None
244     self.blank_icon = None
245     self.update_stage = UPDATE_EVAL
246     self.toggle_counter = 0
247     self.selection_timer = None
248     self.package_selection = None
249     self.cur_pkg = None
250     self.show_all_opts = False
251     self.show_install_updates_only = False
252     self.do_refresh = False
253     self.ua_start = 0
254
255     # Progress Dialog
256     self.gladefile = self.application_dir + \
257         "/usr/share/update-manager/updatesmanager.glade"
258     w_xmltree_progress = gtk.gladefile.XML(self.gladefile, "progressdial
259     self.w_progress_dialog = w_xmltree_progress.get_widget("progress

```

```

260 self.w_progressinfo_label = w_xmltree_progress.get_widget("progr
261 self.w_progressinfo_separator = w_xmltree_progress.get_widget(\
262     "progressinfo_separator")
263 self.w_progressinfo_expander = \
264     w_xmltree_progress.get_widget("progressinfo_expander")
265 self.w_progressinfo_textview = \
266     w_xmltree_progress.get_widget("progressinfo_textview")
267 infobuffer = self.w_progressinfo_textview.get_buffer()
268 infobuffer.create_tag("bold", weight=pango.WEIGHT_BOLD)

270 self.w_progressinfo_expander_label = \
271     w_xmltree_progress.get_widget("progressinfo_expander_lab
272
273 self.w_progress_install_vbox = \
274     w_xmltree_progress.get_widget("progress_install_vbox")
275
276 self.w_progress_eval_img = \
277     w_xmltree_progress.get_widget("progress_eval_img")
278 self.w_progress_eval_label = \
279     w_xmltree_progress.get_widget("progress_eval_label")
280 self.w_progress_download_img = \
281     w_xmltree_progress.get_widget("progress_download_img")
282 self.w_progress_download_label = \
283     w_xmltree_progress.get_widget("progress_download_label")
284 self.w_progress_install_img = \
285     w_xmltree_progress.get_widget("progress_install_img")
286 self.w_progress_install_label = \
287     w_xmltree_progress.get_widget("progress_install_label")
288 self.w_progress_index_img = \
289     w_xmltree_progress.get_widget("progress_index_img")
290 self.w_progress_index_label = \
291     w_xmltree_progress.get_widget("progress_index_label")
292 self.w_progress_closeon_finish_chk = \
293     w_xmltree_progress.get_widget("closeon_finish_checkbutto

295 self.w_progress_cancel = w_xmltree_progress.get_widget("progress
296 self.w_progress_ok = w_xmltree_progress.get_widget("progressok")
297 self.w_progressbar = w_xmltree_progress.get_widget("progressbar")
298
299 # UM Dialog
300 w_xmltree_um = gtk.glade.XML(self.gladefile, "um_dialog")
301 self.w_um_dialog = w_xmltree_um.get_widget("um_dialog")
302
303 self.w_um_dialog.connect("destroy", self.__on_um_dialog_close)
304 self.w_um_intro_label = w_xmltree_um.get_widget("um_intro_label")
305 self.w_um_install_button = w_xmltree_um.get_widget("um_install_b
306 self.w_um_updateall_button = \
307     w_xmltree_um.get_widget("um_updateall_button")
308 self.w_um_expander = w_xmltree_um.get_widget("um_expander")
309 self.w_um_expander.set_expanded(True)

311 self.w_progress_dialog.set_transient_for(self.w_um_dialog)

313 self.w_um_scrolledwindow = w_xmltree_um.get_widget("um_scrolledw
314 self.w_um_treeview = w_xmltree_um.get_widget("um_treeview")
315 self.w_um_textview = w_xmltree_um.get_widget("um_textview")
316 infobuffer = self.w_um_textview.get_buffer()
317 infobuffer.create_tag("bold", weight=pango.WEIGHT_BOLD)
318 self.w_select_checkbox = w_xmltree_um.get_widget("selectall_chec
319 self.w_delete_button = w_xmltree_um.get_widget("cancel_button")
320
321 # ua_confirm_dialog
322 w_xmltree_ua = gtk.glade.XML(self.gladefile, "ua_confirm_dialog")
323 self.w_ua_confirm_dialog = \
324     w_xmltree_ua.get_widget("ua_confirm_dialog")
325 self.w_ua_be_name_entry = \

```

```

326     w_xmltree_ua.get_widget("ua_be_name_entry")
327 self.w_ua_proceed_button = \
328     w_xmltree_ua.get_widget("ua_proceed_button")
329 self.w_ua_cancel_button = \
330     w_xmltree_ua.get_widget("ua_cancel_button")
331
332 self.details_cache = {}
333
334 try:
335     dic = \
336     {
337         "on_um_dialog_close": \
338             self.__on_um_dialog_close,
339         "on_cancel_button_clicked": \
340             self.__on_cancel_button_clicked,
341         "on_install_button_clicked": \
342             self.__on_install_button_clicked,
343         "on_um_updateall_button_clicked": \
344             self.__on_updateall_button_clicked,
345         "on_um_expander_activate": \
346             self.__on_um_expander_activate,
347         "on_selectall_checkbutton_toggled": \
348             self.__on_selectall_checkbutton_toggled,
349     }
350     w_xmltree_um.signal_autoconnect(dic)
351
352     dic_progress = \
353     {
354         "on_progresscancel_clicked": \
355             self.__on_progresscancel_clicked,
356         "on_progressok_clicked": \
357             self.__on_progressok_clicked,
358     }
359     w_xmltree_progress.signal_autoconnect(dic_progress)
360
361     dic_ua = \
362     {
363         "on_ua_cancel_button_clicked": \
364             self.__on_ua_cancel_button_clicked,
365         "on_ua_proceed_button_clicked": \
366             self.__on_ua_proceed_button_clicked,
367     }
368     w_xmltree_ua.signal_autoconnect(dic_ua)
369
370 except AttributeError, error:
371     print self._('GUI will not respond to any event! %s. \
372         Check updatesmanager.py signals') \
373         % error
374
375 self.pr = GUIProgressTracker(self)
376 self.api_obj = None
377
378 self.w_um_dialog.show_all()

380 def __set_cancel_state(self, status):
381     if self.install_error:
382         return
383
384     if status:
385         GObject.idle_add(self.w_progress_cancel.grab_focus)
386
387         GObject.idle_add(self.w_progress_cancel.set_sensitive, status)
388
389 def __progress_step(self, type_step, img, label, str_step):
390     if type_step == UPDATE_ACTIVE:
391         self.__progress_active_step(img, label, str_step)

```

```

392         elif type_step == UPDATE_DONE:
393             self.__progress_done_step(img, label, str_step)
394         else:
395             self.__progress_inactive_step(img, label, str_step)
396
397     def __progress_steps(self, eval_type, evaluate, dl_type, download, insta
398         install, index_type, index):
399         self.__progress_step(eval_type, self.w_progress_eval_img, \
400             self.w_progress_eval_label, evaluate)
401         self.__progress_step(dl_type, self.w_progress_download_img, \
402             self.w_progress_download_label, download)
403         self.__progress_step(install_type, self.w_progress_install_img,
404             self.w_progress_install_label, install)
405         self.__progress_step(index_type, self.w_progress_index_img, \
406             self.w_progress_index_label, index)
407
408     def __progress_steps_start(self):
409         self.__progress_steps(\
410             UPDATE_ACTIVE, self._("Evaluate"), \
411             UPDATE_INACTIVE, self._("Download"), \
412             UPDATE_INACTIVE, self._("Install"), \
413             UPDATE_INACTIVE, self._("Index"))
414
415     def __progress_steps_download(self):
416         self.__progress_steps(\
417             UPDATE_DONE, self._("Evaluate"), \
418             UPDATE_ACTIVE, self._("Download"), \
419             UPDATE_INACTIVE, self._("Install"), \
420             UPDATE_INACTIVE, self._("Index"))
421
422     def __progress_steps_install(self):
423         self.__progress_steps(\
424             UPDATE_DONE, self._("Evaluate"), \
425             UPDATE_DONE, self._("Download"), \
426             UPDATE_ACTIVE, self._("Install"), \
427             UPDATE_INACTIVE, self._("Index"))
428
429     def __progress_steps_index(self):
430         self.__progress_steps(\
431             UPDATE_DONE, self._("Evaluate"), \
432             UPDATE_DONE, self._("Download"), \
433             UPDATE_DONE, self._("Install"), \
434             UPDATE_ACTIVE, self._("Index"))
435
436     def __progress_steps_done(self):
437         self.__progress_steps(\
438             UPDATE_DONE, self._("Evaluate"), \
439             UPDATE_DONE, self._("Download"), \
440             UPDATE_DONE, self._("Install"), \
441             UPDATE_DONE, self._("Index"))
442
443     def __progress_cancel_eval(self):
444         self.__progress_cancel_step(self.w_progress_eval_img, \
445             self.w_progress_eval_label, self._("Evaluate - canceling"))
446
447     def __progress_cancel_download(self):
448         self.__progress_cancel_step(self.w_progress_download_img, \
449             self.w_progress_download_label, self._("Download - cancel"))
450
451
452     def __progress_error_eval(self):
453         self.__progress_error_step(self.w_progress_eval_img, \
454             self.w_progress_eval_label, self._("Evaluate - failed"))
455
456     def __progress_error_download(self):

```

```

458         self.__progress_error_step(self.w_progress_download_img, \
459             self.w_progress_download_label, self._("Download - failed"))
460
461     def __progress_error_install(self):
462         self.__progress_error_step(self.w_progress_install_img, \
463             self.w_progress_install_label, self._("Install - failed"))
464
465     def __progress_error_index(self):
466         self.__progress_error_step(self.w_progress_index_img, \
467             self.w_progress_index_label, self._("Index - failed"))
468
469     @staticmethod
470     def __progress_active_step(widget_image, widget_label, str_step):
471         widget_label.set_markup("<b>%s</b>" % str_step)
472         widget_image.set_from_stock(gtk.STOCK_GO_FORWARD, gtk.ICON_SIZE_
473
474     def __progress_inactive_step(self, widget_image, widget_label, str_step):
475         widget_label.set_text("%s" % str_step)
476         widget_image.set_from_pixbuf(self.blank_icon)
477
478     def __progress_done_step(self, widget_image, widget_label, str_step):
479         widget_label.set_markup("<b>%s</b>" % str_step)
480         widget_image.set_from_pixbuf(self.done_icon)
481
482     def __progress_error_step(self, widget_image, widget_label, str_step):
483         widget_label.set_markup("<b>%s</b>" % str_step)
484         widget_image.set_from_stock(gtk.STOCK_REMOVE, gtk.ICON_SIZE_MENU
485
486         # On error open the Details panel and make sure the Window is vi
487         # to the user, even if it has been minimized
488         self.w_progressinfo_expander.set_expanded(True)
489         self.w_progress_cancel.set_sensitive(True)
490         self.w_um_dialog.present()
491
492     @staticmethod
493     def __progress_cancel_step(widget_image, widget_label, str_step):
494         widget_label.set_markup("<b>%s</b>" % str_step)
495         widget_image.set_from_stock(gtk.STOCK_STOP, gtk.ICON_SIZE_MENU)
496
497     def __set_initial_selection(self):
498         if len(self.um_list) == 0:
499             return
500         self.w_um_treeview.set_cursor(0, None)
501
502     def __remove_installed(self, installed_fmris):
503         model = self.w_um_treeview.get_model()
504         iter_next = model.get_iter_first()
505
506         installed_fmris_dic = dict([(k, None) for k in installed_fmris])
507
508         while iter_next != None:
509             if model.get_value(iter_next, UM_NAME) in installed_fmri
510                 self.um_list.remove(iter_next)
511                 self.toggle_counter -= 1
512             else:
513                 iter_next = model.iter_next(iter_next)
514
515     def __mark_cell_data_default_function(self, column, renderer, model, itr
516         if itr:
517             if model.get_value(itr, UM_STATUS) != None:
518                 self.__set_renderer_active(renderer, False)
519             else:
520                 self.__set_renderer_active(renderer, True)
521
522     @staticmethod

```

```

524 def __set_renderer_active(renderer, active):
525     if active:
526         renderer.set_property("sensitive", True)
527         renderer.set_property("mode", gtk.CELL_RENDERER_MODE_ACT
528     else:
529         renderer.set_property("sensitive", False)
530         renderer.set_property("mode", gtk.CELL_RENDERER_MODE_INE
531
532 def __get_icon_pixbuf(self, icon_name):
533     return self.__get_pixbuf_from_path(self.application_dir + \
534         "/usr/share/icons/update-manager/", icon_name)
535
536 #
537 def get_icon_pixbuf(self, icon_name):
538     return self.__get_pixbuf_from_path(self.application_dir + \
539         "/usr/share/icons/package-manager/", icon_name)
540
541 def __get_app_pixbuf(self, icon_name):
542     return self.__get_pixbuf_from_path(self.application_dir + \
543         "/usr/share/update-manager/", icon_name)
544
545 def __get_pixbuf_from_path(self, path, icon_name):
546     icon = icon_name.replace('.', '_')
547
548     # Performance: Faster to check if files exist rather than catchi
549     # exceptions when they do not. Picked up open failures using dtr
550     png_exists = os.path.exists(self.application_dir + path + icon +
551     svg_exists = os.path.exists(self.application_dir + path + icon +
552
553     if not png_exists and not svg_exists:
554         return None
555     try:
556         return gtk.gdk.pixbuf_new_from_file( \
557             self.application_dir + path + icon + ".png")
558     except GObject.GError:
559         try:
560             return gtk.gdk.pixbuf_new_from_file( \
561                 self.application_dir + path + icon + ".svg")
562         except GObject.GError:
563             iconview = gtk.IconView()
564             icon = iconview.render_icon(getattr(gtk, \
565                 "STOCK_MISSING_IMAGE"), \
566                 size = gtk.ICON_SIZE_MENU,
567                 detail = None)
568             # XXX Could return image-we don't want to show u
569             return None
570
571 def __get_selected_fmris(self):
572     model = self.w_um_treeview.get_model()
573     iter_next = model.get_iter_first()
574     list_of_selected_fmris = []
575     while iter_next != None:
576         if model.get_value(iter_next, UM_INSTALL_MARK):
577             list_of_selected_fmris.append(model.get_value(it
578                 UM_NAME))
579             iter_next = model.iter_next(iter_next)
580     return list_of_selected_fmris
581
582 def init_tree_views(self):
583     model = self.w_um_treeview.get_model()
584     toggle_renderer = gtk.CellRendererToggle()
585     toggle_renderer.connect('toggled', self.__active_pane_toggle, mo
586     column = gtk.TreeViewColumn("", toggle_renderer, \
587         active = UM_INSTALL_MARK)
588     column.set_cell_data_func(toggle_renderer, \

```

```

590         self.__mark_cell_data_default_function, None)
591     toggle_renderer.set_property("activatable", True)
592     column.set_expand(False)
593
594 # Show Cancel and Install Updates + selection column + checkbox
595 if self.show_install_updates_only:
596     self.w_um_updateall_button.hide()
597     self.show_all_opts = True
598
599 # Show Cancel, Update All and Install Updates + selection column
600 if self.show_all_opts:
601     self.w_select_checkbox.show()
602     self.w_um_install_button.show()
603     self.w_um_treeview.append_column(column)
604     self.w_um_intro_label.set_text(self._(\
605         "Updates are available for the following packages.\n" \
606         "Select the packages you want to update and click Instal
607 # Show Cancel, Update All only
608 else:
609     self.w_select_checkbox.hide()
610     self.w_um_install_button.hide()
611     self.w_um_intro_label.set_text(self._(\
612         "Updates are available for the following packages.\n" \
613         "Click Update All to create a new boot environment and "
614         "install all packages into it.))
615
616 render_pixbuf = gtk.CellRendererPixbuf()
617 column = gtk.TreeViewColumn()
618 column.pack_start(render_pixbuf, expand = False)
619 column.add_attribute(render_pixbuf, "pixbuf", UM_STATUS)
620 column.set_title(" ")
621 # Hiding Status column for now
622 #self.w_um_treeview.append_column(column)
623
624 name_renderer = gtk.CellRendererText()
625 column = gtk.TreeViewColumn(self._("Name"), name_renderer, \
626     text = UM_NAME)
627 column.set_cell_data_func(name_renderer, self.__cell_data_func
628 column.set_expand(True)
629 self.w_um_treeview.append_column(column)
630
631 column = gtk.TreeViewColumn()
632 render_pixbuf = gtk.CellRendererPixbuf()
633 column.pack_start(render_pixbuf, expand = True)
634 column.add_attribute(render_pixbuf, "pixbuf", UM_REBOOT)
635 # Hiding Reboot required column for now
636 # self.w_um_treeview.append_column(column)
637
638 version_renderer = gtk.CellRendererText()
639 version_renderer.set_property('xalign', 0.0)
640 column = gtk.TreeViewColumn(self._("Latest Version"), version_re
641     text = UM_LATEST_VER)
642 column.set_cell_data_func(version_renderer, \
643     self.__cell_data_function, None)
644 column.set_expand(True)
645 self.w_um_treeview.append_column(column)
646
647 size_renderer = gtk.CellRendererText()
648 size_renderer.set_property('xalign', 0.0)
649 column = gtk.TreeViewColumn(self._("Size (Meg)"), size_renderer,
650     text = UM_SIZE)
651 column.set_cell_data_func(size_renderer, self.__cell_data_func
652 column.set_expand(True)
653 # XXX Hiding Size as it takes too long to fetch at the minute
654 #self.w_um_treeview.append_column(column)

```

```

656         #Added selection listener
657         self.package_selection = self.w_um_treeview.get_selection()
658         self.package_selection.set_mode(gtk.SELECTION_SINGLE)
659         self.package_selection.connect("changed", \
660             self.__on_package_selection_changed, None)

662         # Setup Icons
663         self.done_icon = self.__get_icon_pixbuf("status_checkmark")
664         self.blank_icon = self.__get_icon_pixbuf("status_blank")
665         self.w_um_dialog.set_icon(self.__get_app_pixbuf("PM_package_36x")
666             self.w_ua_confirm_dialog.set_icon(self.__get_app_pixbuf("PM_pack

668     @staticmethod
669     def __get_image_path():
670         '''This function gets the image path or the default'''
671         if local_image_dir != None:
672             return local_image_dir
673
674         image_directory = os.popen(IMAGE_DIR_COMMAND).readline().rstrip()
675         if len(image_directory) == 0:
676             image_directory = IMAGE_DIRECTORY_DEFAULT
677         return image_directory

679     def get_updates_to_list(self):
680         '''This function fetches a list of the updates
681             that are available to list'''
682         # MUST match self.um_list ListStore setup in __init__
683         um_list = \
684             gtk.ListStore(
685                 GObject.TYPE_INT,           # UM_ID
686                 GObject.TYPE_BOOLEAN,      # UM_INSTALL_MARK
687                 Gdk.Pixbuf,                # UM_STATUS
688                 GObject.TYPE_STRING,       # UM_NAME
689                 Gdk.Pixbuf,                # UM_REBOOT
690                 GObject.TYPE_STRING,       # UM_LATEST_VER
691                 GObject.TYPE_STRING,       # UM_SIZE
692                 GObject.TYPE_STRING,       # UM_FMRI
693             )

695         image_obj = self.__get_image_obj_from_directory(self.__get_image
696
697         count = 0
698         pkg_upgradeable = None
699         for pkg, state in sorted(image_obj.inventory(all_known = True)):
700             while gtk.events_pending():
701                 gtk.main_iteration(False)
702             if state["upgradable"] and state["state"] == "installed"
703                 pkg_upgradeable = pkg
704
705             # Allow testing by listing uninstalled packages, -u opti
706             add_package = False
707             if pkg_upgradeable != None and not state["upgradable"]:
708                 if list_uninstalled:
709                     add_package = not \
710                         image_obj.fmri_is_same_pkg(pkg_upgradeab
711                 else:
712                     add_package = \
713                         image_obj.fmri_is_same_pkg(pkg_upgradeab

715         if add_package:
716             count += 1
717             # XXX: Would like to caputre if package for upgr
718             # incorporated, could then indicate this to user
719             # and take action when doing install to run imag
720             #if state["incorporated"]:
721                 #             incState = self._("Inc")

```

```

722         #else:
723         #             incState = "--"
724
725         um_list.insert(count, [count, False, None, \
726             pkg.get_name(), None, pkg.get_version(),
727             pkg.get_fmri()])
728
729         if debug:
730             print self._("count: %d") % count

732         self.progress_stop_thread = True
733         GObject.idle_add(self.w_um_treeview.set_model, um_list)
734         GObject.idle_add(self.__set_initial_selection)
735         self.um_list = um_list
736         self.__selectall_toggle(True)
737         if len(self.um_list) == 0:
738             self.__display_noupdates()
739             return

741         # XXX: Currently this will fetch the sizes but it really slows d
742         # app responsiveness - until we get caching I think we should ju
743         # the size column
744         # GObject.timeout_add(1000, self.__setup_sizes)

746     def __get_api_obj(self):
747         if self.api_obj != None:
748             return self.api_obj
749
750         try:
751             self.api_obj = api.ImageInterface(self.__get_image_path(
752                 CLIENT_API_VERSION, self.pr, self.__set_cancel_s
753                 PKG_CLIENT_NAME)
754             return self.api_obj
755         except api_errors.ImageNotFoundException, ine:
756             self.w_um_expander.set_expanded(True)
757             infobuffer = self.w_um_textview.get_buffer()
758             infobuffer.set_text("")
759             textiter = infobuffer.get_end_iter()
760             infobuffer.insert_with_tags_by_name(textiter, self._("Er
761                 "bold")
762             infobuffer.insert(textiter, \
763                 self._("%s' is not an install image\n") % \
764                 ine.user_specified)
765         except api_errors.VersionException, ve:
766             self.w_um_expander.set_expanded(True)
767             infobuffer = self.w_um_textview.get_buffer()
768             infobuffer.set_text("")
769             textiter = infobuffer.get_end_iter()
770             infobuffer.insert_with_tags_by_name(textiter, self._("Er
771                 "bold")
772             infobuffer.insert(textiter,
773                 self._("Version mismatch: expected %s received %
774                 (ve.expected_version, ve.received_version))
775             return None

777     def __display_noupdates(self):
778         self.w_um_scrolledwindow.set_policy(gtk.POLICY_NEVER, \
779             gtk.POLICY_AUTOMATIC)
780         self.w_um_expander.set_expanded(True)
781         infobuffer = self.w_um_textview.get_buffer()
782         textiter = infobuffer.get_end_iter()
783         infobuffer.insert_with_tags_by_name(textiter, \
784             "\nNo Updates available", "bold")

785         self.w_um_install_button.set_sensitive(False)
786         self.w_um_updateall_button.set_sensitive(False)
787         self.w_select_checkbox.set_active(False)

```

```

788         self.w_select_checkbox.set_sensitive(False)
789         self.w_um_dialog.present()
790
791     def __get_info_from_name(self, name):
792         local = True
793         get_license = False
794
795         if self.fmri_description != name:
796             return None
797         if self.__get_api_obj() == None:
798             return None
799
800         ret = self.__get_api_obj().info([name], local, get_license)
801
802         pis = ret[api.ImageInterface.INFO_FOUND]
803         if len(pis) == 1:
804             return pis[0]
805         else:
806             return None
807
808     def __get_details_from_name(self, name):
809         info = self.__get_info_from_name(name)
810         if info is not None:
811             return self.__update_details_from_info(name, info)
812         else:
813             return None
814
815     def __update_details_from_info(self, name, info):
816         ver = "%s-%s" % (info.version, info.branch)
817         str_details = self._(\
818             '\nDescription:\t\t%s\nFMRI:           \t\t%s' + \
819             '\nVersion:           \t\t%s\nPackaged on:\t\t%s' + \
820             '\nSize:             \t\t\t%.3f MB\n') \
821             % (info.summary, info.fmri, ver, info.packaging_date, \
822                info.size / 1024.0 / 1024.0)
823         self.details_cache[name] = str_details
824         return str_details
825
826     # This is copied from a similar function in packagemanager.py
827     def __get_image_obj_from_directory(self, image_directory):
828         image_obj = image.Image()
829         dr = "/"
830         try:
831             image_obj.find_root(image_directory)
832             while gtk.events_pending():
833                 gtk.main_iteration(False)
834             image_obj.load_config()
835             while gtk.events_pending():
836                 gtk.main_iteration(False)
837             image_obj.load_catalogs(self.pr)
838             while gtk.events_pending():
839                 gtk.main_iteration(False)
840         except ValueError:
841             print self._('%s is not valid image, trying root image')
842                 % image_directory
843             try:
844                 dr = os.environ["PKG_IMAGE"]
845             except KeyError:
846                 print
847             try:
848                 image_obj.find_root(dr)
849                 image_obj.load_config()
850             except ValueError:
851                 print self._('%s is not valid root image, return
852                     % dr
853                 image_obj = None

```

```

854         return image_obj
855
856     @staticmethod
857     def __removed_filter(model, itr):
858         '''This function filters category in the main application view'''
859         return True
860
861     def __on_package_selection_changed(self, selection, widget):
862         '''This function is for handling package selection changes'''
863         model, itr = selection.get_selected()
864         if itr:
865             fmri = model.get_value(itr, UM_NAME)
866             delta = time.time() - self.last_select_time
867             if delta < SELECTION_CHANGE_LIMIT:
868                 if self.selection_timer is not None:
869                     self.selection_timer.cancel()
870                     self.selection_timer = None
871
872             self.fmri_description = fmri
873             self.last_select_time = time.time()
874
875             if self.details_cache.has_key(fmri):
876                 if self.selection_timer is not None:
877                     self.selection_timer.cancel()
878                     self.selection_timer = None
879                 infobuffer = self.w_um_textview.get_buffer()
880                 infobuffer.set_text("")
881                 textiter = infobuffer.get_end_iter()
882                 infobuffer.insert_with_tags_by_name(textiter, \
883                     "\n%s\n" % fmri, "bold")
884                 infobuffer.insert(textiter, self.details_cache[fmri])
885             else:
886                 infobuffer = self.w_um_textview.get_buffer()
887                 infobuffer.set_text(\
888                     self._("\nFetching details for %s ...")
889                     self.selection_timer = Timer(SELECTION_CHANGE_LI
890                     self.__show_package_info_thread, \
891                     args=(fmri,)).start())
892
893     def __show_package_info_thread(self, fmri):
894         Thread(target = self.__show_package_info, \
895             args = (fmri,)).start()
896
897     def __show_package_info(self, fmri):
898         details = self.__get_details_from_name(fmri)
899         if self.fmri_description == fmri and details != None:
900             infobuffer = self.w_um_textview.get_buffer()
901             infobuffer.set_text("")
902             textiter = infobuffer.get_end_iter()
903             infobuffer.insert_with_tags_by_name(textiter, \
904                 "\n%s\n" % fmri, "bold")
905             infobuffer.insert(textiter, details)
906         elif self.fmri_description == fmri and details == None:
907             infobuffer = self.w_um_textview.get_buffer()
908             textiter = infobuffer.get_end_iter()
909             infobuffer.insert_with_tags_by_name(textiter, \
910                 "\nNo details available", "bold")
911
912     def __on_um_dialog_close(self, widget):
913         self.__exit_app()
914
915     def __on_cancel_button_clicked(self, widget):
916         self.__exit_app()
917
918     def __exit_app(self):

```

```

920         self.cancelled = True
921         gtk.main_quit()
922         sys.exit(0)
923         return True
924
925     def __on_progressok_clicked(self, widget):
926         self.w_progress_dialog.hide()
927
928     def __on_progresscancel_clicked(self, widget):
929         if self.install_error:
930             self.w_progress_dialog.hide()
931             self.w_progressinfo_expander.set_expanded(False)
932
933         if self.api_obj != None and self.api_obj.can_be_canceled():
934             if self.update_stage == UPDATE_EVAL:
935                 self.__progress_cancel_eval()
936             elif self.update_stage == UPDATE_DOWNLOAD:
937                 self.__progress_cancel_download()
938
939             self.__update_progress_info(\
940                 self._("\nCanceling update, please wait ..."))
941             self.w_progress_cancel.set_sensitive(False)
942             Thread(target = self.api_obj.cancel).start()
943         else:
944             self.__update_progress_info(\
945                 self._("\nUnable to cancel at this time."))
946
947     def __on_install_button_clicked(self, widget):
948         self.setup_progressdialog_show(self._("Installing Updates"), \
949             showCancel = True, showOK = True, \
950             isInstall = True, showCloseOnFinish = debug)
951         Thread(target = self.__install).start()
952
953     def __on_ua_cancel_button_clicked(self, widget):
954         self.w_ua_confirm_dialog.hide()
955         return
956
957     def __on_ua_proceed_button_clicked(self, widget):
958         self.w_ua_confirm_dialog.hide()
959         self.ua_start = time.time()
960         self.setup_progressdialog_show(self._("Update All"), \
961             showCancel = True, showOK = True, \
962             isInstall = True, showCloseOnFinish = debug)
963
964         Thread(target = self.__update_image, \
965             args = (self.w_ua_be_name_entry.get_text(),)).start()
966         return
967
968     def __on_updateall_button_clicked(self, widget):
969         self.__selectall_toggle(True)
970         date_str = time.strftime("%m/%d/%Y", time.localtime())
971         self.w_ua_be_name_entry.set_text("opensolaris-ua-%s" % date_str)
972         self.w_ua_proceed_button.grab_focus()
973         self.w_ua_confirm_dialog.show()
974         return
975
976     def __on_selectall_checkbutton_toggled(self, widget):
977         self.__selectall_toggle(widget.get_active())
978
979     def __handle_incorporated_error(self, list_incorp):
980         self.__update_progress_info(self._("ERROR"), True)
981         self.__update_progress_info(\
982             self._("Following Incorporated package(s) cannot be upda
983         for i in list_incorp:
984             self.__update_progress_info("\t%s" % i)
985         self.__update_progress_info(\

```

```

986         self._("Update using: Update All\n"), True)
987
988     def __handle_update_progress_error(self, str_error, ex = None, \
989         stage = UPDATE_EVAL):
990         self.install_error = True
991         if stage == UPDATE_EVAL:
992             gobject.idle_add(self.__progress_error_eval)
993         elif stage == UPDATE_DOWNLOAD:
994             gobject.idle_add(self.__progress_error_download)
995         elif stage == UPDATE_INSTALL:
996             gobject.idle_add(self.__progress_error_install)
997         elif stage == UPDATE_INDEX:
998             gobject.idle_add(self.__progress_error_index)
999         else:
1000             gobject.idle_add(self.__progress_error_eval)
1001
1002         gobject.idle_add(self.__update_progress_info,
1003             self._("\nERROR"), True)
1004         if ex != None:
1005             gobject.idle_add(self.__update_progress_info, \
1006                 self._("%s\n%s" % (str_error, ex)))
1007         else:
1008             gobject.idle_add(self.__update_progress_info, \
1009                 self._("%s\n" % str_error))
1010         self.__cleanup()
1011
1012     def __update_image(self, be_name = None):
1013         self.install = True
1014         self.install_error = False
1015         self.update_stage = UPDATE_EVAL
1016
1017         # Evaluate
1018         try:
1019             gobject.idle_add(self.__update_progress_info, \
1020                 self._("\nEvaluate\n"), True)
1021             if self.__get_api_obj() == None:
1022                 return
1023
1024             stuff_to_do, opensolaris_image, cre = \
1025                 self.__get_api_obj().plan_update_all(sys.argv[0],
1026                 refresh_catalogs = self.do_refresh)
1027             #XXX waiting for change to API to allow be name to b
1028             # self.api_obj.plan_update_all(sys.argv[0], be_name)
1029             if cre:
1030                 self.__handle_update_progress_error(\
1031                     self._(
1032                         "Update All failed during catalog refres
1033                         "while determining what to update:"), cr
1034                     stage = self.update_stage)
1035                 return
1036             if not opensolaris_image:
1037                 self.__handle_update_progress_error(\
1038                     self._("This is not an opensolaris image
1039                     stage = self.update_stage)
1040                 return
1041             if not stuff_to_do:
1042                 self.__handle_update_progress_error(\
1043                     self._("No updates available for this im
1044                     stage = self.update_stage)
1045                 return
1046         except (api_errors.CanceledException):
1047             self.__handle_cancel_exception()
1048             return
1049         except api_errors.CatalogRefreshException, cre:
1050             self.__handle_update_progress_error(\
1051                 self._("Update All failed during catalog refresh

```

```

1052         "while determining what to update:)", cre, \
1053         stage = self.update_stage)
1054     except api_errors.PlanCreationException, pce:
1055         self.__handle_update_progress_error(\
1056             self._("Update All failure in plan creation:"),
1057             stage = self.update_stage)
1058     return
1059     except api_errors.IpkgOutOfDateException:
1060         self.__handle_update_progress_error(\
1061             self._(\
1062                 "pkg(5) appears to be out of date and should be\
1063                 updated before running Update All.\n" + \
1064                 "Please update pkg(5) using:\n" + \
1065                 "\t'pfexec pkg install SUNWipkg' " + \
1066                 "and then retry Update All."), \
1067             stage = self.update_stage)
1068     return
1069     except api_errors.ApiException, aex:
1070         self.__handle_update_progress_error(\
1071             self._("Update All API failure in evaluation:"),
1072             stage = self.update_stage)
1073     return
1074     except Exception, uex:
1075         self.__handle_update_progress_error(\
1076             self._("Update All unexpected error in evaluatio
1077             uex, stage = self.update_stage)
1078     return
1079
1080     if self.__shared_update_steps(self._("Update All"), \
1081         self._("Update All finished successfully.\n")) != 0:
1082         return
1083
1084     gobject.idle_add(self.__display_update_image_success)
1085
1086 def __display_update_image_success(self):
1087     elapsed = (time.time() - self.ua_start) / 60.0
1088     info_str = ""
1089     if elapsed >= 1.0:
1090         info_str = \
1091             self._(\
1092                 "\nUpdate All finished successfully in %1.f mins
1093                 elapsed)
1094     else:
1095         info_str = \
1096             self._(\
1097                 "\nUpdate All finished successfully in < 1 min\n
1098
1099     info_str += self._(\
1100         "Please reboot after reviewing the release notes posted
1101         "http://opensolaris.org/os/project/indiana/resources/" \
1102         "relnotes/200811/x86/")
1103
1104     self.w_um_dialog.hide()
1105     msgbox = gtk.MessageDialog(parent = self.w_um_dialog, \
1106         buttons = gtk.BUTTONS_CLOSE, flags = gtk.DIALOG_MODAL, \
1107         type = gtk.MESSAGE_INFO, \
1108         message_format = info_str)
1109     msgbox.set_title(self._("Update All Completed"))
1110     msgbox.run()
1111     msgbox.destroy()
1112     self.__exit_app()
1113
1114 def __handle_cancel_exception(self):
1115     gobject.idle_add(self.w_progress_dialog.hide)
1116     gobject.idle_add(self.w_progressinfo_expander.set_expanded, Fals
1117     self.__cleanup()

```

```

1119     def __install(self):
1120         self.install = True
1121         self.install_error = False
1122         self.update_stage = UPDATE_EVAL
1123         list_fmris_to_install = self.__get_selected_fmris()
1124         if len(list_fmris_to_install) == 0:
1125             self.__handle_update_progress_error(\
1126                 self._("Nothing selected to update.))
1127         return
1128
1129     if self.__get_api_obj() == None:
1130         return
1131
1132     if debug:
1133         print self._("Updating ...")
1134         print list_fmris_to_install
1135
1136     # Evaluate
1137     try:
1138         gobject.idle_add(self.__update_progress_info, \
1139             self._("\nEvaluate\n"), True)
1140         ret, exception_caught = \
1141             self.__get_api_obj().plan_install(list_fmris_to
1142             [], refresh_catalogs = self.do_refresh)
1143         if exception_caught != None:
1144             self.__handle_update_progress_error(\
1145                 self._("Update error in plan install:"),
1146                 exception_caught, \
1147                 stage = self.update_stage)
1148         return
1149
1150     except (api_errors.CanceledException):
1151         self.__handle_cancel_exception()
1152         return
1153     except (api_errors.ApiException), aex:
1154         self.__handle_update_progress_error(\
1155             self._("Update unexpected API error:"), aex, \
1156             stage = self.update_stage)
1157         return
1158     except (Exception), uex:
1159         self.__handle_update_progress_error(\
1160             self._("Update unexpected error:"), uex, \
1161             stage = self.update_stage)
1162         return
1163
1164     if not ret:
1165         #XXX Nothing to do, must be an incorporated package
1166         # need to offer Update All to user
1167         self.install_error = True
1168         gobject.idle_add(self.__progress_error_eval)
1169         gobject.idle_add(self.__handle_incorporated_error, \
1170             list_fmris_to_install)
1171         self.__cleanup()
1172         return
1173
1174     list_changes = self.__get_api_obj().describe().get_changes()
1175     list_planned = [x[1].pkg_stem for x in list_changes]
1176
1177     if len(list_planned) != len(list_fmris_to_install):
1178         list_incorp = self.__unique(list_fmris_to_install, list_
1179         gobject.idle_add(self.__progress_error_eval)
1180         gobject.idle_add(self.__handle_incorporated_error, list_
1181
1182     gobject.idle_add(self.__update_progress_info, \
1183         self._("Packages to be installed:"))

```

```

1184         for i in list_planned:
1185             gobject.idle_add(self.__update_progress_info, "\t%s" % i

1187     if self.__shared_update_steps(self._("Update"), \
1188         self._("Update finished successfully.")) != 0:
1189         return
1190
1191     gobject.idle_add(self.__remove_installed, list_planned)
1192     gobject.idle_add(self.w_um_install_button.set_sensitive, False)
1193     gobject.idle_add(self.w_progressinfo_expander.set_expanded, False)
1194
1195     def __shared_update_steps(self, what_msg, success_msg):
1196         # Download
1197         try:
1198             self.update_stage = UPDATE_DOWNLOAD
1199             self.__get_api_obj().prepare()
1200         except (api_errors.CanceledException):
1201             self.__handle_cancel_exception()
1202             return 1
1203         except (api_errors.ApiException), aex:
1204             self.install_error = True
1205             gobject.idle_add(self.__progress_error_download)
1206             gobject.idle_add(self.__update_progress_info, \
1207                 self._("\nERROR"), True)
1208             gobject.idle_add(self.__update_progress_info, \
1209                 self._("%s Download failed:\n%s" % (what_msg, aex
1210                     self.__cleanup())
1211                 return 1
1212         except EnvironmentError, uex:
1213             if uex.errno in (errno.EDQUOT, errno.ENOSPC):
1214                 self.__handle_update_progress_error(
1215                     self._(
1216                         "%s exceded available disc space" % (what
1217                     except (Exception), uex:
1218                         if uex.args[0] == errno.EDQUOT or uex.args[0] == errno.E
1219                             self.__handle_update_progress_error(\
1220                                 self._(\
1221                                     "%s exceded available disc space" % (what
1222                                     stage = self.update_stage)
1223                                 gobject.idle_add(self.__prompt_to_load_beadm)
1224                             else:
1225                                 self.__handle_update_progress_error(
1226                                     self._("%s unexpected error:" % (what_ms
1227                                     uex, stage = self.update_stage)
1228                                 return 1
1229
1230         # Install
1231         try:
1232             self.update_stage = UPDATE_INSTALL
1233             gobject.idle_add(self.w_progress_cancel.set_sensitive, False)
1234             self.__get_api_obj().execute_plan()
1235         except (api_errors.CanceledException):
1236             self.__handle_cancel_exception()
1237             return 1
1238         except (api_errors.ApiException), aex:
1239             self.install_error = True
1240             gobject.idle_add(self.__progress_error_install)
1241             gobject.idle_add(self.__update_progress_info, \
1242                 self._("\nERROR"), True)

```

```

1243     gobject.idle_add(self.__update_progress_info, \
1244         self._("%s Execute plan failed:\n%s" % (what_msg
1245         self.__cleanup()
1246         return 1
1247     except EnvironmentError, uex:
1248         if uex.errno in (errno.EDQUOT, errno.ENOSPC):
1249             self.__handle_update_progress_error(
1250                 self._(
1251                     "%s exceded available disc space" % (what
1252             except (Exception), uex:
1253                 if uex.args[0] == errno.EDQUOT or uex.args[0] == errno.E
1254                     self.__handle_update_progress_error(\
1255                         self._(\
1256                             "%s exceded available disc space" % (what
1257                             stage = self.update_stage)
1258                 gobject.idle_add(self.__prompt_to_load_beadm)
1259             else:
1260                 self.__handle_update_progress_error(
1261                     self._("%s unexpected error:" % (what_ms
1262                     uex, stage = self.update_stage)
1263                 return 1
1264             return 1
1265         except Exception, uex:
1266             self.__handle_update_progress_error(
1267                 self._("%s unexpected error:" % (what_msg)),
1268                 self.__handle_update_progress_error(\
1269                     self._("%s unexpected error:" % (what_ms
1270                     uex, stage = self.update_stage)
1271             return 1
1272
1273     self.__cleanup()
1274     gobject.idle_add(self.__progress_steps_done)
1275     gobject.idle_add(self.__update_progress_info, \
1276         self._(success_msg), True)
1277     gobject.idle_add(self.w_progress_ok.set_sensitive, True)
1278
1279     return 0
1280
1281     def __prompt_to_load_beadm(self):
1282     msgbox = gtk.MessageDialog(parent = self.w_progress_dialog, \
1283         buttons = gtk.BUTTONS_OK_CANCEL, flags = gtk.DIALOG_MODAL
1284         type = gtk.MESSAGE_ERROR, \
1285         message_format = self._(\
1286             "Not enough disc space: the Update All action cannot " \
1287             "be performed.\n\n" \
1288             "Click OK to launch BE Management to manage your " \
1289             "existing BE's and free up disc space.")
1290     msgbox.set_title(self._("Not Enough Disc Space"))
1291     result = msgbox.run()
1292     msgbox.destroy()
1293     if result == gtk.RESPONSE_OK:
1294         gobject.idle_add(self.__create_beadm)
1295
1296     def __create_beadm(self):
1297         self.gladefile = \
1298             "/usr/share/package-manager/packagemanager.glade"
1299         beadm.Beadm(self)
1300         return False
1301
1302     @staticmethod
1303     def __unique(list1, list2):
1304         """Return a list containing all items
1305             in 'list1' that are not in 'list2'"""
1306         list2 = dict([(k, None) for k in list2])
1307         return [item for item in list1 if item not in list2]
1308
1309     def __cleanup(self):

```

```

1302         self.install = False
1303         self.api_obj.reset()
1304         self.pr.reset()
1305         self.progress_stop_thread = True
1306
1307     @staticmethod
1308     def __on_um_expander_activate(widget):
1309         return
1310
1311     def __selectall_toggle(self, select):
1312         for row in self.um_list:
1313             row[UM_INSTALL_MARK] = select
1314         if select:
1315             self.toggle_counter += len(self.um_list)
1316             self.w_um_install_button.set_sensitive(True)
1317         else:
1318             self.toggle_counter = 0
1319             self.w_um_install_button.set_sensitive(False)
1320
1321     def __active_pane_toggle(self, cell, filtered_path, filtered_model):
1322         model = self.w_um_treeview.get_model()
1323         itr = model.get_iter(filtered_path)
1324         if itr:
1325             installed = model.get_value(itr, UM_STATUS)
1326             if installed is None:
1327                 modified = model.get_value(itr, UM_INSTALL_MARK)
1328                 model.set_value(itr, UM_INSTALL_MARK, not modified)
1329             if not modified:
1330                 self.toggle_counter += 1
1331             else:
1332                 self.toggle_counter -= 1
1333
1334         if self.toggle_counter > 0:
1335             self.w_um_install_button.set_sensitive(True)
1336         else:
1337             self.toggle_counter = 0
1338             self.w_um_install_button.set_sensitive(False)
1339
1340     @staticmethod
1341     def __cell_data_function(column, renderer, model, itr, data):
1342         '''Function which sets the background colour to black if package
1343         selected'''
1344         if itr:
1345             if model.get_value(itr, 1):
1346                 #XXX Setting BOLD looks too noisy - disable for
1347                 # renderer.set_property("weight", pango.WEIGHT_B
1348                 renderer.set_property("weight", pango.WEIGHT_NOR
1349             else:
1350                 renderer.set_property("weight", pango.WEIGHT_NOR
1351
1352     def __on_progressdialog_progress(self, isInstall):
1353         if not self.progress_stop_thread:
1354             self.w_progressbar.pulse()
1355             return True
1356         else:
1357             if isInstall:
1358                 self.w_progressbar.set_fraction(0.0)
1359                 if not self.install_error and \
1360                     self.w_progress_closeon_finish_chk.get_a
1361                     self.w_progress_dialog.hide()
1362             else:
1363                 self.w_progress_dialog.hide()
1364             return False
1365
1366     def setup_progressdialog_show(self, title, info = None, showDetails = Tr
1367         showCancel = False, showOK = False, isInstall = False, \

```

```

1368         showCloseOnFinish = False):
1369         infobuffer = self.w_progressinfo_textview.get_buffer()
1370         infobuffer.set_text("")
1371
1372         if info != None:
1373             self.w_progressinfo_label.set_text(info)
1374             self.w_progressinfo_label.show()
1375         else:
1376             self.w_progressinfo_label.hide()
1377
1378         if showDetails:
1379             self.w_progressinfo_expander.show()
1380             self.w_progressinfo_separator.show()
1381             self.w_progressinfo_expander.set_expanded(False)
1382         else:
1383             self.w_progressinfo_expander.hide()
1384             self.w_progressinfo_separator.hide()
1385
1386         if showCancel:
1387             self.w_progress_cancel.show()
1388             self.w_progress_cancel.set_sensitive(True)
1389             self.w_progress_cancel.grab_focus()
1390         else:
1391             self.w_progress_cancel.hide()
1392
1393         if showOK:
1394             self.w_progress_ok.show()
1395             self.w_progress_ok.set_sensitive(False)
1396         else:
1397             self.w_progress_ok.hide()
1398
1399         if isInstall:
1400             self.__progress_steps_start()
1401             self.w_progress_install_vbox.show()
1402         else:
1403             self.w_progress_install_vbox.hide()
1404
1405         if showCloseOnFinish:
1406             self.w_progress_closeon_finish_chk.show()
1407             self.w_progress_ok.show()
1408         else:
1409             self.w_progress_closeon_finish_chk.hide()
1410             self.w_progress_ok.hide()
1411
1412         self.w_progress_dialog.set_title(title)
1413
1414         self.w_progress_dialog.show()
1415         self.progress_stop_thread = False
1416         GObject.timeout_add(100, self.__on_progressdialog_progress, isIn
1417
1418     def setup_updates(self):
1419         Thread(target = self.get_updates_to_list(), args = []).start()
1420         return False
1421
1422     @staticmethod
1423     def __update_size(size, pkg):
1424         pkg[UM_SIZE] = size/ 1024.0 /1024.0 # Display in MB
1425
1426     # Handle GUI Progress Output
1427     def output(self, str_out):
1428         self.__update_progress_info(str_out)
1429         if debug:
1430             print str_out
1431
1432     def output_done(self, what="not specified"):
1433         self.__update_progress_info(" ")

```

```

1434
1435         if debug:
1436             print "%s: finished" % what
1437
1438     @staticmethod
1439     def progress_pulse():
1440         if debug:
1441             print "pulse: \n"
1442
1443     def dl_progress(self, dl_started, dl_cur_pkg, \
1444                   dl_cur_npkgs, dl_goal_npkgs, \
1445                   dl_cur_nfiles, dl_goal_nfiles, \
1446                   dl_cur_nmegbytes, dl_goal_nmegbytes):
1447         if not dl_started:
1448             self.cur_pkg = ""
1449             self.__progress_steps_download()
1450             self.__update_progress_info(self._("\nDownload\n"), True)
1451
1452         if self.cur_pkg != dl_cur_pkg:
1453             self.__update_progress_info("%s" % dl_cur_pkg)
1454         self.cur_pkg = dl_cur_pkg
1455
1456         self.__update_progress_info(\
1457             self._("\tPKG %d/%d: \tfiles %d/%d \tXfer %.2f/%.2f(meg)
1458                 (dl_cur_npkgs, dl_goal_npkgs, \
1459                 dl_cur_nfiles, dl_goal_nfiles, \
1460                 dl_cur_nmegbytes, dl_goal_nmegbytes))
1461
1462         if debug:
1463             print "DL: %s - %s\nPKG %d/%d: files %d/%d: megs %.2f/%.
1464                   (dl_started, dl_cur_pkg, \
1465                   dl_cur_npkgs, dl_goal_npkgs, \
1466                   dl_cur_nfiles, dl_goal_nfiles, \
1467                   dl_cur_nmegbytes, dl_goal_nmegbytes)
1468
1469     def act_progress(self, act_started, \
1470                    act_phase, act_cur_nactions, act_goal_nactions):
1471
1472         if not act_started:
1473             self.__progress_steps_install()
1474             self.__update_progress_info(self._("\nInstall\n"), True)
1475             self.__update_progress_info(
1476                 self._("\t%s\t%d/%d actions") % \
1477                     (act_phase, act_cur_nactions, act_goal_nactions))
1478         else:
1479             self.__update_progress_info(
1480                 self._("\t%s\t%d/%d actions") % \
1481                     (act_phase, act_cur_nactions, act_goal_nactions))
1482
1483         if debug:
1484             print "Install: %s - %s\nACT %d/%d\n" % \
1485                   (act_started, act_phase, act_cur_nactions, \
1486                   act_goal_nactions)
1487
1488     def ind_progress(self, ind_started, \
1489                    ind_phase, ind_cur_nitems, ind_goal_nitems):
1490         if not self.install:
1491             return
1492
1493         if not ind_started:
1494             if self.update_stage != UPDATE_INSTALL:
1495                 return
1496             self.update_stage = UPDATE_INDEX
1497             self.__progress_steps_index()
1498             self.__update_progress_info(self._("Index\n"), True)
1499             self.__update_progress_info(self._("\t%-25s\t%d/%d actio

```

```

1500                                     (ind_phase, ind_cur_nitems, ind_goal_nitems)))
1501         else:
1502             self.__update_progress_info(
1503                 self._("\t%-25s\t%d/%d actions") % \
1504                     (ind_phase, ind_cur_nitems, ind_goal_nitems))
1505
1506         if debug:
1507             print "Index: %s - %s\nACT %d/%d\n" % \
1508                   (ind_started, ind_phase, ind_cur_nitems, ind_goa
1509
1510     def __update_progress_info(self, str_out, bold = False):
1511         infobuffer = self.w_progressinfo_textview.get_buffer()
1512         textiter = infobuffer.get_end_iter()
1513
1514         # Requires TextView tag to be setup once in __init__
1515         # infobuffer.create_tag("bold", weight=pango.WEIGHT_BOLD)
1516         if bold:
1517             infobuffer.insert_with_tags_by_name(textiter, \
1518                 "%s\n" % str_out, "bold")
1519         else:
1520             infobuffer.insert(textiter, "%s\n" % str_out)
1521         self.w_progressinfo_textview.scroll_to_iter(textiter, 0.0)
1522
1523 #----- remove those
1524 def main():
1525     gtk.main()
1526     return 0
1527
1528 if __name__ == "__main__":
1529     list_uninstalled = False
1530     debug = False
1531     show_all_opts = False
1532     show_install_updates_only = False
1533     local_image_dir = None
1534     do_refresh = False
1535
1536     try:
1537         opts, args = getopt.getopt(sys.argv[1:], "h dualir", \
1538                                   ["help", "debug", "uninstalled"])
1539     except getopt.error, msg:
1540         print "%s, for help use --help" % msg
1541         sys.exit(2)
1542
1543     for option, argument in opts:
1544         if option in ("-h", "--help"):
1545             print "Use -d (--debug) to run in debug mode."
1546             sys.exit(0)
1547         if option in ("-d", "--debug"):
1548             debug = True
1549         if option in ("-u", "--uninstalled"):
1550             list_uninstalled = True
1551         if option in ("-a", "--all"):
1552             show_all_opts = True
1553         if option in ("-i", "--install_updates"):
1554             show_install_updates_only = True
1555         if option in ("-l", "--local_image"):
1556             local_image_dir = os.getcwd()
1557         # Refresh catalogs during plan_install and plan_update_all
1558         if option in ("-r", "--refresh"):
1559             do_refresh = True
1560
1561     um = Updatemanager()
1562     um.show_all_opts = show_all_opts
1563     um.show_install_updates_only = show_install_updates_only
1564     um.do_refresh = do_refresh
1565     um.init_tree_views()

```

```
1567     um.setup_progressdialog_show(um._("Checking for new software"), \  
1568         showDetails = False)  
1569     gobject.timeout_add(UPDATES_FETCH_DELAY, um.setup_updates)  
1570     main()
```