

```

*****
48364 Fri Jul 3 10:40:32 2009
new/src/gui/modules/installupdate.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 2009 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #

26 MIN_IND_ELEMENTS_BOUNCE = 5      # During indexing the progress will be progress
27                                 # the number of indexing elements is greater th
28                                 # otherwise it will bounce

30 import errno
31 import os
32 import sys
33 import time
34 import pango
35 import datetime
36 import traceback
37 import string
38 from threading import Thread
39 try:
40     import gobject
41     import gtk
42     import gtk.glade
43     import pygtk
44     pygtk.require("2.0")
45 except ImportError:
46     sys.exit(1)
47 nobe = False
48 try:
49     import libbe as be
50 except ImportError:
51     nobe = True
52 import pkg.client.progress as progress
53 import pkg.misc
54 import pkg.client.api_errors as api_errors
55 import pkg.gui.beadmin as beadm
56 import pkg.gui.misc as gui_misc
57 import pkg.gui.enumerations as enumerations

59 ERROR_FORMAT = "<span color = \"red\">%s</span>"

```

```

62 class InstallUpdate(progress.ProgressTracker):
63     def __init__(self, list_of_packages, parent, api_o,
64                 ips_update = False, action = -1, be_name = None,
65                 parent_name = "", pkg_list = None, main_window = None,
66                 icon_confirm_dialog = None, title = None, web_install = False):
67         if action == -1:
68             return
69         progress.ProgressTracker.__init__(self)
70         self.web_install = web_install
71         self.web_updates_list = None
72         self.parent = parent
73         self.be_name = be_name
74         self.parent_name = parent_name
75         self.ipkg_ipkgui_list = pkg_list
76         self.icon_confirm_dialog = icon_confirm_dialog
77         self.title = title
78         self.w_main_window = main_window
79         self.ips_update = ips_update
80         self.list_of_packages = list_of_packages
81         self.act_phase_last = None
82         self.action = action
83         self.canceling = False
84         self.current_stage_name = None
85         self.ind_started = None
86         self.ip = None
87         self.operations_done = False
88         self.prev_ind_phase = None
89         self.prev_pkg = None
90         self.progress_stop_timer_running = False
91         self.proposed_be_name = None
92         self.stages = {
93             1: [_(("Preparing...")), _(("Preparation"))],
94             2: [_(("Downloading...")), _(("Download"))],
95             3: [_(("Installing...")), _(("Install"))],
96         }
97
98         self.stop_bouncing_progress = False
99         self.stopped_bouncing_progress = True
100         self.update_list = {}
101         gladefile = os.path.join(self.parent.application_dir,
102                                 "usr/share/package-manager/packagemanager.glade")
103         w_tree_dialog = gtk.glade.XML(gladefile, "createplndialog")
104         w_tree_uconfirm = gtk.glade.XML(gladefile, "ua_confirm_dialog")
105         w_tree_removeconfirm = \
106             gtk.glade.XML(gladefile, "removeconfirmation")
107         self.w_dialog = w_tree_dialog.get_widget("createplndialog")
108         self.w_expander = w_tree_dialog.get_widget("expander3")
109         self.w_cancel_button = w_tree_dialog.get_widget("cancelcreatepla")
110         self.w_progressbar = w_tree_dialog.get_widget("createplanprogres")
111         self.w_details_textview = w_tree_dialog.get_widget("createplante")
112         self.w_removeconfirm_dialog = \
113             w_tree_removeconfirm.get_widget("removeconfirmation")
114         w_removeproceed_button = w_tree_removeconfirm.get_widget("remove")
115         w_remove_treeview = w_tree_removeconfirm.get_widget("removetreev")
116         w_stage2 = w_tree_dialog.get_widget("stage2")
117         self.w_stages_box = w_tree_dialog.get_widget("stages_box")
118         self.w_stage1_label = w_tree_dialog.get_widget("label_stage1")
119         self.w_stage1_icon = w_tree_dialog.get_widget("icon_stage1")
120         self.w_stage2_label = w_tree_dialog.get_widget("label_stage2")
121         self.w_stage2_icon = w_tree_dialog.get_widget("icon_stage2")
122         self.w_stage3_label = w_tree_dialog.get_widget("label_stage3")
123         self.w_stage3_icon = w_tree_dialog.get_widget("icon_stage3")
124         self.w_stages3_label = w_tree_dialog.get_widget("label_stages")
125         self.w_stages3_icon = w_tree_dialog.get_widget("icon_stages")
126         self.w_stages_label = w_tree_dialog.get_widget("label_stages")
127         self.w_stages_icon = w_tree_dialog.get_widget("icon_stages")

```

```

128     self.current_stage_label = self.w_stagel_label
129     self.current_stage_icon = self.w_stagel_icon
130     self.current_stage_label_done = None

132     self.done_icon = gui_misc.get_icon(
133         self.parent.icon_theme, "progress_checkmark")
134     blank_icon = gui_misc.get_icon(
135         self.parent.icon_theme, "progress_blank")

137     self.w_stagel_icon.set_from_pixbuf(blank_icon)
138     self.w_stage2_icon.set_from_pixbuf(blank_icon)
139     self.w_stage3_icon.set_from_pixbuf(blank_icon)

141     infobuffer = self.w_details_textview.get_buffer()
142     infobuffer.create_tag("bold", weight=pango.WEIGHT_BOLD)
143     infobuffer.create_tag("level1", left_margin=30, right_margin=10)
144     infobuffer.create_tag("level2", left_margin=50, right_margin=10)
145     self.w_ua_dialog = w_tree_uaconfirm.get_widget("ua_confirm_dialo
146     self.w_ua_error_label = w_tree_uaconfirm.get_widget(
147         "ua_confirm_error_label")
148     self.w_ua_proceed_button = w_tree_uaconfirm.get_widget(
149         "ua_proceed_button")
150     self.w_ua_be_name_entry = w_tree_uaconfirm.get_widget(
151         "ua_be_name_entry")
152     self.w_ua_be_name_box = w_tree_uaconfirm.get_widget(
153         "ua_be_name_box")

155     w_ua_proceed_button = w_tree_uaconfirm.get_widget("ua_proceed_bu
156     self.w_progressbar.set_pulse_step(0.02)
157     try:
158         dic_createplan = \
159         {
160             "on_cancelcreateplan_clicked": \
161                 self.__on_cancelcreateplan_clicked,
162             "on_createplandialog_delete_event": \
163                 self.__on_createplandialog_delete,
164         }
165         dic_uaconfirm = \
166         {
167             "on_ua_cancel_button_clicked": \
168                 self.__on_ua_cancel_button_clicked,
169             "on_ua_proceed_button_clicked": \
170                 self.__on_ua_proceed_button_clicked,
171             "on_ua_be_name_entry_changed": \
172                 self.__on_ua_be_name_entry_changed,
173             "on_ua_help_button_clicked": \
174                 self.__on_ua_help_button_clicked,
175         }
176         dic_removeconfirm = \
177         {
178             "on_proceed_button_clicked": \
179                 self.__on_remove_proceed_button_clicked,
180             "on_cancel_button_clicked": \
181                 self.__on_remove_cancel_button_clicked,
182         }
183         w_tree_dialog.signal_autoconnect(dic_createplan)
184         w_tree_uaconfirm.signal_autoconnect(dic_uaconfirm)
185         w_tree_removeconfirm.signal_autoconnect(dic_removeconfir
186     except AttributeError, error:
187         print _("GUI will not respond to any event! %s. "
188             "Check installupdate.py signals") \
189             % error

192     self.w_dialog.set_transient_for(self.w_main_window)
193     self.w_ua_dialog.set_transient_for(self.w_main_window)

```

```

194     if self.icon_confirm_dialog != None:
195         self.w_ua_dialog.set_icon(self.icon_confirm_dialog)

197     if self.action == enumerations.REMOVE:
198         #We are not showing the download stage in the main stage
199         self.stages[3] = [_(("Removing...")), _(("Remove"))]
200         self.w_stage3_label.set_text(self.stages[3][1])
201         w_stage2.hide()
202         self.w_dialog.set_title(_("Remove"))
203         w_removeproceed_button.grab_focus()
204         cell = gtk.CellRendererText()
205         remove_column = gtk.TreeViewColumn('Removed')
206         remove_column.pack_start(cell, True)
207         remove_column.add_attribute(cell, 'text', 0)
208         w_remove_treeview.append_column(remove_column)

210         liststore = gtk.ListStore(str)
211         for sel_pkg in list_of_packages:
212             liststore.append([sel_pkg])
213         w_remove_treeview.set_model(liststore)
214         w_remove_treeview.expand_all()
215         self.w_removeconfirm_dialog.show()

217     elif self.action == enumerations.IMAGE_UPDATE:
218         self.w_dialog.set_title(_("Update All"))
219         w_ua_proceed_button.grab_focus()
220         if not self.be_name:
221             if nobe or not "beVerifyBEName" in be.__dict__:
222                 self.w_ua_be_name_box.set_property(
223                     "visible", False)
224             else:
225                 self.__setup_be_list()
226                 self.w_ua_dialog.show()
227         else:
228             self.proposed_be_name = self.be_name
229             self.__proceed_with_stages()

230     else:
231         if self.title != None:
232             self.w_dialog.set_title(self.title)
233         else:
234             self.w_dialog.set_title(_("Install/Update"))
235         self.__proceed_with_stages()

238     def __on_createplandialog_delete(self, widget, event):
239         self.__on_cancelcreateplan_clicked(None)
240         return True

242     def __on_cancelcreateplan_clicked(self, widget):
243         '''Handler for signal send by cancel button, which user might pr
244         evaluation stage - while the dialog is creating plan'''
245         if self.api_o.can_be_canceled():
246             self.canceling = True
247             Thread(target = self.api_o.cancel, args = ()).start()
248             cancel_txt = _("Canceling...")
249             txt = "<b>" + self.current_stage_label_done + " - " \
250                 + cancel_txt + "</b>"
251             gobject.idle_add(self.current_stage_label.set_markup, tx
252             gobject.idle_add(self.current_stage_icon.set_from_stock,
253                 gtk.STOCK_CANCEL, gtk.ICON_SIZE_MENU)
254             gobject.idle_add(self.w_stages_label.set_markup, cancel_
255             self.w_cancel_button.set_sensitive(False)
256         if self.operations_done:
257             self.w_dialog.hide()
258         if self.web_install:
259             gobject.idle_add(self.parent.update_package_list

```

```

260         self.web_updates_list)
261     return
262     gobject.idle_add(self.parent.update_package_list, None)

264 def __on_ua_help_button_clicked(self, widget):
265     gui_misc.display_help(self.parent.application_dir, "update_all")
266
267 def __on_ua_cancel_button_clicked(self, widget):
268     self.w_ua_dialog.hide()
269     if self.web_install:
270         gobject.idle_add(self.parent.update_package_list,
271                         self.web_updates_list)
272     return
273     gobject.idle_add(self.parent.update_package_list, None)

275 def __on_ua_proceed_button_clicked(self, widget):
276     proposed_be_name = self.w_ua_be_name_entry.get_text()
277     if proposed_be_name != "":
278         self.proposed_be_name = proposed_be_name
279     self.w_ua_dialog.hide()
280     self.__proceed_with_stages()

282 def __setup_be_list(self):
283     be_list = be.beList()
284     error_code = None
285     if len(be_list) > 1 and type(be_list[0]) == type(-1):
286         error_code = be_list[0]
287     if error_code != None and error_code == 0:
288         self.be_list = be_list[1]
289     elif error_code == None:
290         self.be_list = be_list

292     # Now set proposed name in entry field.
293     active_name = None
294     for bee in self.be_list:
295         name = bee.get("orig_be_name")
296         if name:
297             if bee.get("active"):
298                 active_name = name
299                 break

300     if active_name != None:
301         proposed_name = None
302         list = string.rsplit(active_name, '-', 1)
303         if len(list) == 1:
304             proposed_name = self.__construct_be_name(
305                 active_name, 0)
306         else:
307             try:
308                 i = int(list[1])
309                 proposed_name = self.__construct_be_name(
310                     list[0], i)
311             except ValueError:
312                 proposed_name = self.__construct_be_name(
313                     active_name, 0)

315     if proposed_name != None:
316         self.w_ua_be_name_entry.set_text(proposed_name)

318 def __construct_be_name(self, name, i):
319     in_use = True
320     proposed_name = None
321     while in_use:
322         i += 1
323         proposed_name = name + '-' + str(i)
324         in_use = self.__is_be_name_in_use(proposed_name)

```

```

326         return proposed_name
327
328 def __is_be_name_in_use(self, name):
329     in_use = False
330     if name == "":
331         return in_use
332     for bee in self.be_list:
333         be_name = bee.get("orig_be_name")
334         if be_name == name:
335             in_use = True
336             break
337     return in_use
338
339 def __is_be_name_valid(self, name):
340     if name == "":
341         return True
342     return be.beVerifyBEName(name) == 0
343
344 def __validate_be_name(self, widget):
345     name = widget.get_text()
346     is_name_valid = self.__is_be_name_valid(name)
347     self.w_ua_error_label.hide()
348     error_str = None
349     if is_name_valid:
350         is_name_in_use = self.__is_be_name_in_use(name)
351         if is_name_in_use:
352             error_str = ERROR_FORMAT % _("BE name is in use")
353     else:
354         error_str = ERROR_FORMAT % _("BE name is invalid")
355     if error_str != None:
356         self.w_ua_error_label.set_markup(error_str)
357         self.w_ua_error_label.show()
358
359     self.w_ua_proceed_button.set_sensitive(error_str == None)

361 def __on_ua_be_name_entry_changed(self, widget):
362     self.__validate_be_name(widget)

364 def __on_remove_cancel_button_clicked(self, widget):
365     self.w_removeconfirm_dialog.hide()

367 def __on_remove_proceed_button_clicked(self, widget):
368     self.w_removeconfirm_dialog.hide()
369     self.__proceed_with_stages()

371 def __ipkg_ipkgui_uptodate(self):
372     if self.ipkg_ipkgui_list == None:
373         return True
374     upgrade_needed, cre = self.api_o.plan_install(
375         self.ipkg_ipkgui_list, filters = [])
376     return not upgrade_needed

378 def __proceed_with_stages(self):
379     self.__start_stage_one()
380     self.w_dialog.show()
381     Thread(target = self.__proceed_with_stages_thread_ex,
382            args = []).start()

384 def __proceed_with_stages_thread_ex(self):
385     try:
386         if self.action == enumerations.IMAGE_UPDATE:
387             self.__start_substage(
388                 _("Ensuring %s is up to date...") % self.par
389                 bounce_progress=True)
390             opensolaris_image = True
391             ips_uptodate = True

```

```

392         notfound = self.__installed_fmris_from_args(
393             ["SUNWipkg", "SUNWcs"])
394         if notfound:
395             opensolaris_image = False
396         if opensolaris_image:
397             ips_uptodate = self.__ipkg_ipkgui_uptoda
398         if not ips_uptodate:
399             #Do the stuff with installing ipkg ipkkg
400             #restart in the special mode
401             self.ips_update = True
402             self.__proceed_with_ipkg_thread()
403             return
404         else:
405             self.api_o.reset()
406             self.__proceed_with_stages_thread()
407     except api_errors.CertificateError, e:
408         self.__g_error_stage(str(e))
409     except api_errors.CertificateError:
410         self.stop_bouncing_progress = True
411         msg = _("Accessing this restricted repository failed."
412             "\nYou either need to register to access this reposi
413             "\nthe certificate expired, or you need to accept th
414             "\nrepository\ncertificate.")
415         self.__g_error_stage(msg)
416     return
417 except api_errors.PlanCreationException, e:
418     self.__g_error_stage(str(e))
419     return
420 except api_errors.InventoryException, e:
421     msg = _("Inventory exception:\n")
422     if e.illegal:
423         for i in e.illegal:
424             msg += "\tpkg:\t" + i + "\n"
425     self.__g_error_stage(msg)
426     return
427 except api_errors.CatalogRefreshException, e:
428     msg = _("Please check the network "
429         "connection.\nIs the repository accessible?")
430     if e.message and len(e.message) > 0:
431         msg = e.message
432     self.__g_error_stage(msg)
433     return
434 except api_errors.TransportError, ex:
435     msg = _("Please check the network "
436         "connection.\nIs the repository accessible?\n\n"
437         "%s") % str(ex)
438     self.__g_error_stage(msg)
439     return
440 except api_errors.InvalidDepotResponseException, e:
441     msg = _("Unable to contact a valid package depot. "
442         "Please check your network\nsettings and "
443         "attempt to contact the server using a web "
444         "browser.\n\n%s") % str(e)
445     self.__g_error_stage(msg)
446     return
447 except api_errors.IpkgOutOfDateException:
448     msg = _("pkg(5) appears to be out of "
449         "date and should be\nupdated before running "
450         "Update All.\nPlease update SUNWipkg package")
451     self.__g_error_stage(msg)
452     return
453 except api_errors.NonLeafPackageException, nlpe:
454     msg = _("Cannot remove:\n\t%s\n"
455         "Due to the following packages that "
456         "depend on it:\n") % nlpe[0].get_name()
457     for pkg_a in nlpe[1]:

```

```

451         msg += "\t" + pkg_a.get_name() + "\n"
452     self.__g_error_stage(msg)
453     return
454 except api_errors.ProblematicPermissionsIndexException, err:
455     msg = str(err)
456     msg += _("\nFailure of consistent use of pfexec or gksu
457     "running\n%s is often a source of this problem.") %
458     self.parent_name
459     msg += _("\nTo rebuild index, please use the terminal co
460     msg += _("\n\tpfexec pkg rebuild-index")
461     self.__g_error_stage(msg)
462     return
463 except api_errors.CorrupedIndexException:
464     msg = _("There was an error during installation. The sea
465     "index is corrupted. You might want try to fix this\
466     "problem by running command:\n"
467     "\tpfexec pkg rebuild-index")
468     self.__g_error_stage(msg)
469     return
470 except api_errors.ImageUpdateOnLiveImageException:
471     msg = _("This is an Live Image. The install"
472     "\noperation can't be performed.")
473     self.__g_error_stage(msg)
474     return
475 except api_errors.PlanMissingException:
476     msg = _("There was an error during installation.\n"
477     "The Plan of the operation is missing and the operat
478     "can't be finished. You might want try to fix this\n"
479     "problem by restarting %s\n") % self.parent_name
480     self.__g_error_stage(msg)
481     return
482 except api_errors.ImageplanStateException:
483     msg = _("There was an error during installation.\n"
484     "The State of the image is incorrect and the operati
485     "can't be finished. You might want try to fix this\n"
486     "problem by restarting %s\n") % self.parent_name
487     self.__g_error_stage(msg)
488     return
489 except api_errors.CanceledException:
490     gobject.idle_add(self.w_dialog.hide)
491     self.stop_bouncing_progress = True
492     return
493 except api_errors.BENamingNotSupported:
494     msg = _("Specifying BE Name not supported.\n")
495     self.__g_error_stage(msg)
496     return
497 except api_errors.InvalidBENAMEException:
498     msg = _("Invalid BE Name: %s.\n") % self.proposed_be_nam
499     self.__g_error_stage(msg)
500     return
501 except api_errors.PermissionsException, pex:
502     msg = str(pex)
503     self.__g_error_stage(msg)
504     return
505 except (api_errors.UnableToCopyBE,
506     api_errors.UnableToMountBE,
507     api_errors.BENAMEGivenOnDeadBE,
508     api_errors.UnableToRenameBE), ex:
509     msg = str(ex)
510     self.__g_error_stage(msg)
511     return
512 except Exception, uex:
513     # We do want to prompt user to load BE admin if there is
514     # not enough disk space. This error can either come as a
515     # error within API exception, see bug #7642 or as a stan
516     # error, that is why we need to check for both situation

```



```

649         self.__g_update_details_text("%s\n" % traceback, "level2")
650     else:
651         msg = _("No further information available")
652         self.__g_update_details_text("%s\n" % msg, "level2")
653     gobject.idle_add(self.w_expander.set_expanded, True)
654     gobject.idle_add(self.w_cancel_button.set_sensitive, True)

656 def __start_substage(self, text, bounce_progress=True):
657     if text:
658         gobject.idle_add(self.__stages_label.set_markup, text)
659         self.__g_update_details_text(text + "\n")
660     if bounce_progress:
661         if self.stopped_bouncing_progress:
662             self.__start_bouncing_progress()
663     else:
664         self.stop_bouncing_progress = True

666 def __stages_label_set_markup(self, markup_text):
667     if not self.canceling == True:
668         self.w_stages_label.set_markup(markup_text)

670 def __start_bouncing_progress(self):
671     self.stop_bouncing_progress = False
672     self.stopped_bouncing_progress = False
673     Thread(target =
674            self.__g_progressdialog_progress_pulse).start()

676 def __g_progressdialog_progress_pulse(self):
677     while not self.stop_bouncing_progress:
678         gobject.idle_add(self.w_progressbar.pulse)
679         time.sleep(0.1)
680     self.stopped_bouncing_progress = True

682 def __g_update_details_text(self, text, *tags):
683     gobject.idle_add(self.__update_details_text, text, *tags)

685 def __update_details_text(self, text, *tags):
686     buf = self.w_details_textview.get_buffer()
687     textiter = buf.get_end_iter()
688     if tags:
689         buf.insert_with_tags_by_name(textiter, text, *tags)
690     else:
691         buf.insert(textiter, text)
692     self.w_details_textview.scroll_to_iter(textiter, 0.0)

694 def __update_download_progress(self, cur_bytes, total_bytes):
695     prog = float(cur_bytes)/total_bytes
696     self.w_progressbar.set_fraction(prog)
697     size_a_str = ""
698     size_b_str = ""
699     if cur_bytes >= 0:
700         size_a_str = pkg.misc.bytes_to_str(cur_bytes)
701     if total_bytes >= 0:
702         size_b_str = pkg.misc.bytes_to_str(total_bytes)
703     c = _("Downloaded %(current)s of %(total)s") % \
704         {"current" : size_a_str,
705          "total" : size_b_str}
706     self.__stages_label_set_markup(c)

708 def __update_install_progress(self, current, total):
709     prog = float(current)/total
710     self.w_progressbar.set_fraction(prog)

712 def __plan_stage(self):
713     '''Function which plans the image'''
714     stuff_to_do = False

```

```

715     if self.action == enumerations.INSTALL_UPDATE:
716         stuff_to_do, cre = self.api_o.plan_install(
717             self.list_of_packages, refresh_catalogs = False,
718             filters = [])
719         if cre and not cre.succeeded:
720             # cre is either None or a catalog refresh except
721             # which was caught while planning.
722             raise api_errors.CatalogRefreshException(None, N
723             ,_("Catalog refresh failed during install."))
724     elif self.action == enumerations.REMOVE:
725         plan_uninstall = self.api_o.plan_uninstall
726         stuff_to_do = \
727             plan_uninstall(self.list_of_packages, False, False)
728     elif self.action == enumerations.IMAGE_UPDATE:
729         # we are passing force, since we already checked if the
730         # SUNWipkg and SUNWipkg-gui are up to date.
731         stuff_to_do, opensolaris_image, cre = \
732             self.api_o.plan_update_all(sys.argv[0],
733             refresh_catalogs = False,
734             noexecute = False, force = True,
735             be_name = self.proposed_be_name)
736         if cre and not cre.succeeded:
737             raise api_errors.CatalogRefreshException(None, N
738             ,_("Catalog refresh failed during Update All
739             return stuff_to_do

741 def __operations_done(self, alternate_done_txt = None):
742     done_txt = _("Installation completed successfully.")
743     if self.action == enumerations.REMOVE:
744         done_txt = _("Packages removed successfully.")
745     elif self.action == enumerations.IMAGE_UPDATE:
746         done_txt = _("Packages updated successfully.")
747     if alternate_done_txt != None:
748         done_txt = alternate_done_txt
749     self.w_stages_box.hide()
750     self.w_stages_icon.set_from_stock(
751         gtk.STOCK_OK, gtk.ICON_SIZE_DND)
752     self.w_stages_icon.show()
753     self.__stages_label_set_markup(done_txt)
754     self.__update_details_text("\n"+ done_txt, "bold")
755     self.w_cancel_button.set_label("gtk-close")
756     self.w_progressbar.hide()
757     self.stop_bouncing_progress = True
758     self.operations_done = True
759     if self.parent != None:
760         if not self.web_install and not self.ips_update \
761             and not self.action == enumerations.IMAGE_UPDATE:
762             self.parent.update_package_list(self.update_list
763             if self.web_install:
764                 self.web_updates_list = self.update_list
765         if self.ips_update:
766             self.w_dialog.hide()
767             self.parent.restart_after_ips_update(self.proposed_be_na
768         elif self.action == enumerations.IMAGE_UPDATE:
769             self.w_dialog.hide()
770             self.parent.shutdown_after_image_update()

772 def __prompt_to_load_beadm(self):
773     msgbox = gtk.MessageDialog(parent = self.w_main_window,
774     buttons = gtk.BUTTONS_OK_CANCEL, flags = gtk.DIALOG_MODAL,
775     type = gtk.MESSAGE_ERROR,
776     message_format = _("
777     "Not enough disk space, the selected action cannot "
778     "be performed.\n\n"
779     "Click OK to manage your existing BEs and free up disk s
780     "Cancel to cancel the action."))

```

```

781         msgbox.set_title(_("Not Enough Disk Space"))
782         result = msgbox.run()
783         msgbox.destroy()
784         if result == gtk.RESPONSE_OK:
785             beadm.Beadmin(self.parent)

787     def __afterplan_information(self):
788         install_iter = None
789         update_iter = None
790         remove_iter = None
791         plan = self.api_o.describe().get_changes()
792         self.__g_update_details_text("\n")
793         for pkg_plan in plan:
794             origin_fmri = pkg_plan[0]
795             destination_fmri = pkg_plan[1]
796             if origin_fmri and destination_fmri:
797                 if not update_iter:
798                     update_iter = True
799                     txt = _("Packages To Be Updated:\n")
800                     self.__g_update_details_text(txt, "bold")
801                     pkg_a = self.__get_pkgstr_from_pkginfo(destination_fmri)
802                     self.__g_update_details_text(pkg_a+"\n", "level1")
803             elif not origin_fmri and destination_fmri:
804                 if not install_iter:
805                     install_iter = True
806                     txt = _("Packages To Be Installed:\n")
807                     self.__g_update_details_text(txt, "bold")
808                     pkg_a = self.__get_pkgstr_from_pkginfo(destination_fmri)
809                     self.__g_update_details_text(pkg_a+"\n", "level1")
810             elif origin_fmri and not destination_fmri:
811                 if not remove_iter:
812                     remove_iter = True
813                     txt = _("Packages To Be Removed:\n")
814                     self.__g_update_details_text(txt, "bold")
815                     pkg_a = self.__get_pkgstr_from_pkginfo(origin_fmri)
816                     self.__g_update_details_text(pkg_a+"\n", "level1")
817             self.__g_update_details_text("\n")

819     def __get_pkgstr_from_pkginfo(self, pkginfo):
820         dt_str = self.get_datetime(pkginfo.packaging_date)
821         if not dt_str:
822             dt_str = ""
823         s_ver = pkginfo.version
824         s_bran = pkginfo.branch
825         pkg_name = pkginfo.pkg_stem
826         pkg_publisher = pkginfo.publisher
827         if not pkg_publisher in self.update_list:
828             self.update_list[pkg_publisher] = []
829         pub_list = self.update_list.get(pkg_publisher)
830         if not pkg_name in pub_list:
831             pub_list.append(pkg_name)
832         l_ver = 0
833         version_pref = ""
834         while l_ver < len(s_ver) - 1:
835             version_pref += "%d%s" % (s_ver[l_ver], ".")
836             l_ver += 1
837         version_pref += "%d%s" % (s_ver[l_ver], "-")
838         l_ver = 0
839         version_suf = ""
840         if s_bran != None:
841             while l_ver < len(s_bran) - 1:
842                 version_suf += "%d%s" % (s_bran[l_ver], ".")
843                 l_ver += 1
844             version_suf += "%d" % s_bran[l_ver]
845         pkg_version = version_pref + version_suf + dt_str
846         return pkg_name + "@" + pkg_version

```

```

848     def act_output(self):
849         if self.act_phase != self.act_phase_last:
850             self.act_phase_last = self.act_phase
851             GObject.idle_add(self.__stages_label_set_markup, self.act_phase)
852             self.__g_update_details_text("\n")
853             GObject.idle_add(self.__update_install_progress, self.act_phase)
854             self.act_cur_nactions, self.act_goal_nactions)
855         return

857     def act_output_done(self):
858         return

860     def cat_output_start(self):
861         return

863     def cat_output_done(self):
864         return

866     def cache_cats_output_start(self):
867         return

869     def cache_cats_output_done(self):
870         return

872     def load_cat_cache_output_start(self):
873         return

875     def load_cat_cache_output_done(self):
876         return

878     def dl_output(self):
879         GObject.idle_add(self.__update_download_progress, \
880             self.dl_cur_nbytes, self.dl_goal_nbytes)
881         if self.prev_pkg != self.dl_cur_pkg:
882             self.prev_pkg = self.dl_cur_pkg
883             self.__g_update_details_text(
884                 _("Package %d of %d: %s\n") % (self.dl_cur_npkgs+1,
885                 self.dl_goal_npkgs, self.dl_cur_pkg), "level1")

887     def dl_output_done(self):
888         self.__g_update_details_text("\n")

890     def eval_output_start(self):
891         '''Called by progress tracker when the evaluation of the package
892         started.'''
893         return

895     def eval_output_progress(self):
896         '''Called by progress tracker each time some package was evaluated
897         call is being done by calling progress tracker evaluate_progress
898         function'''
899         if self.prev_pkg != self.eval_cur_fmri:
900             self.prev_pkg = self.eval_cur_fmri
901             self.__g_update_details_text("\n")
902             self.__g_update_details_text(
903                 _("Evaluating: %s") % self.eval_cur_fmri.get_name(),
904                 GObject.idle_add(self.__stages_label_set_markup, text))

906     def eval_output_done(self):
907         return

909     def ind_output(self):
910         if self.ind_started != self.ind_phase:
911             self.ind_started = self.ind_phase
912             GObject.idle_add(self.__stages_label_set_markup, self.in

```

```
913         self.__g_update_details_text(
914             _("%s\n") % (self.ind_phase), "level1")
915         gobject.idle_add(self.__indexing_progress)

917     def __indexing_progress(self):
918         #It doesn't look nice if the progressive is just for few element
919         if self.ind_goal_nitems > MIN_IND_ELEMENTS_BOUNCE:
920             gobject.idle_add(self.__update_install_progress,
921                             self.ind_cur_nitems-1, self.ind_goal_nitems)
922         else:
923             if self.stopped_bouncing_progress:
924                 self.__start_bouncing_progress()

926     def ind_output_done(self):
927         gobject.idle_add(self.__update_install_progress, self.ind_cur_ni
928             self.ind_goal_nitems)

930     def ver_output(self):
931         return

933     def ver_output_error(self, actname, errors):
934         return

936     @staticmethod
937     def get_datetime(date_time):
938         '''Support function for getting date from the API.'''
939         date_tmp = None
940         try:
941             date_tmp = time.strptime(date_time, "%a %b %d %H:%M:%S %
942         except ValueError:
943             return None
944         if date_tmp:
945             date_tmp2 = datetime.datetime(*date_tmp[0:5])
946             return date_tmp2.strftime(":%m%d")
947         return None

949     def __installed_fmris_from_args(self, args_f):
950         found = []
951         notfound = []
952         try:
953             for m in self.api_o.img.inventory(args_f):
954                 found.append(m[0])
955         except api_errors.InventoryException, e:
956             notfound = e.notfound
957         return notfound
```