TkPane Documentation

Release 1.0.0

Dreas Nielsen

Jul 21, 2018

Contents:

1 Concepts of Pane Interactions **3**
- 1.1 Actions . . . 3
- 1.2 Data Sharing . . . 4
- 1.3 Callback Lists . . . 5

2 Usage **7**

3 Interface **9**

4 Creating Custom Panes by Subclassing TkPane **11**

5 Validating Data **13**

6 Setting TkPane Frame Styles **15**

7 Highlighting Invalid Data **17**

8 Changing Widget Types in `tkpane.lib` **19**

9 Package Contents **21**
- 9.1 The TkPane Class . . . 21
- 9.2 Action Method Handling Classes (CbHandler Classes) . . . 26
- 9.3 Functions . . . 26
- 9.4 The Pane Library . . . 27

10 Examples **49**
- 10.1 Example 1: Using Panes from `tkpane.lib` . . . 49
- 10.2 Example 2. Construction of a Custom Pane . . . 51
- 10.3 Example 3. A Simple CSV Explorer Application . . . 54

11 Notes **59**

12 Availability **61**

13 Related Software **63**

14 Copyright and License **65**

15 Contributors **67**

16 Change Log 69

17 Index 71

Python Module Index 73

TkPane is a Python package designed to simplify the construction of a Tkinter user interface by encapsulating one or more widgets into 'pane' objects. Pane objects are independent in that each pane has no inherent dependence on any other pane, but dependencies and data sharing between panes can be established with as little as a single method call. Panes interact with one another through a standardized interface of methods and callback functions. Each pane maintains a dictionary of its own data, and panes can send data to, or request data from, other panes. Other application code can also easily obtain a pane's data in the form of a Python dictionary.

The `tkpane.lib` package includes a number of pre-defined general-purpose panes that may be useful in many user interfaces. The TkPane class in the `tkapne` package is the foundation for custom panes that can be created and tailored to the specific needs of a particular application.

The following figure illustrates a user interface containing four separate panes. Borders have been added to the panes so that they are easily distinguished for the purpose of this illustration. Both input and output file names are required but currently invalid (missing), as indicated by the light red background of the entry widgets. Entry of output comments is dependent on the specification of an output file, and so is currently disabled. The "OK" button is dependent on entry of both an input and output file, and so is also currently disabled.

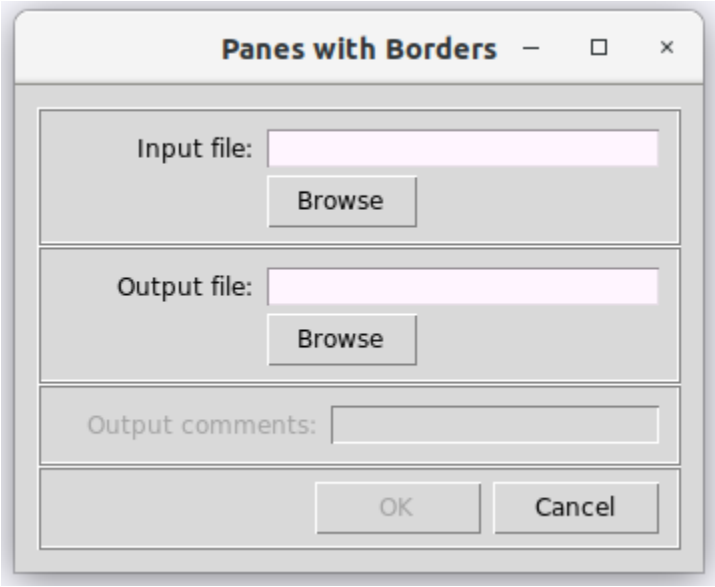

Fig. 1: Figure 1. Four Panes with Borders Added

Each of these dependencies is established by a single method call. The same method calls ensure that, when the "OK" button is enabled, the button pane's internal data will include the input and output filenames and the comment; no other application-level code is required for the button pane to reference any other panes to obtain their data.

CHAPTER 1

Concepts of Pane Interactions

There are two types of interactions between UI elements that are handled by panes created with the `tkpane` package:

- Actions, whereby one pane changes the state of another pane.
- Data sharing, whereby one pane sends data to, or requests data from, another pane.

1.1 Actions

Tkinter widgets, and therefore also the panes created with the `tkpane` package, can have three actions applied to them:

- *Enable*: The pane is made able to accept user input.
- *Disable*: The pane is made unable to accept user input.
- *Clear*: Any data displayed by the pane is removed.

These actions are not applicable to all panes. For example a pane that simply displays a text message using a Tkinter Label widget does not ordinarily accept any kind of user input, so these actions do not apply. A uniform interface is established for all panes, however, so even a pane containing only a Label widget will allow these actions to be applied—they simply will not do anything (although a custom pane could be created containing a Label widget that does respond in some way to these actions).

Conversely, some panes must support additional actions. For example, a pane containing a Text widget must support actions that change the text, and a pane containing buttons must support actions that allow each button's functionality to be specified.

The *enable*, *disable*, and *clear* actions, however, are the core actions needed for panes to interact with each other. Changes that take place in one pane–changes made to the pane's data by the user, or actions that are applied to the pane—may cause that pane to enable, disable, or clear one or more other panes. The `tkpane` package encapsulates those inter-pane dependencies so that they are easily specified and automatically carried out. For example,

```
pane_b.requires(pane_a)
```

uses the `requires()` method of a pane object to specify that one pane (`pane_b`) will not be enabled until another pane (`pane_a`) contains valid data. The `requires()` method will ensure that `pane_a` disables `pane_b` if its own data are invalid, and enables `pane_b` when its data become valid. If `pane_a` is cleared by a call to it's `clear()` method, `pane_b`'s data will be automatically cleared as well. The *clear* action can also be carried out when `pane_b` is enabled or disabled, with a command like

```
pane_b.requires(pane_a, clear_on_enable=True)
```

or

```
pane_b.requires(pane_a, clear_on_disable=True)
```

These options to the `requires()` method will clear the data from `pane_b` as part of the specified action.

In addition to the *clear* action that removes any visible data from the pane, and clears the corresponding value from the internal data dictionary, a related action is to remove some or all stored data *other than* the data managed by the pane itself. This *clear data* action is appropriate when one pane will use another pane's data if that other pane contains valid data, but the first pane does not require the second pane's data. The `can_use()` method represents this type of relationship between panes:

```
pane_c.can_use(pane_a)
```

Thereafter, if the data on `pane_a` become valid, `pane_a` will send `pane_c` its own data dictionary, and when data on `pane_a` become invalid, `pane_a` will call `pane_c`'s `clear_data()` method, passing it `pane_a`'s data keys.

1.2 Data Sharing

Each pane that manages data—a pane that has an Entry widget to collect user input, for example—maintains its internal data in a Python dictionary. That dictionary only contains valid data, so if no data has been entered, the dictionary may be empty. Each pane maintains not only the data dictionary itself, but also a separate list of all of the valid dictionary keys for data that is managed by the pane.

Each pane's data dictionary can store not only the pane's own data, but also any data that have been provided by, or acquired from, other panes. For example, a pane containing an "OK" button may not be enabled until valid data have been entered on two or three other panes, and for the "OK" button's action to be carried out, the data from those other panes is needed. This type of data sharing is facilitated by the *enable* action. When one pane enables another pane (calls the `enable()` method of the second pane), it passes its own data dictionary. The pane that is enabled can store, use, or ignore those data as appropriate. If the pane that is enabled is an "OK" button, for example, it may store those data in its own data dictionary to be used when its own *OK* action is carried out.

One pane can also request the data managed by another pane using the `values()` method of the second pane. This will return a dictionary containing only the data managed by that pane, not including any data that may have been obtained by that pane from other panes. This data dictionary may be empty or incomplete if the data on the providing pane are not valid. The `valid_data()` method can be used to check whether or not a pane's data are valid.

One pane can also request all of the data stored in another pane's data dictionary using the `all_data()` method of the second pane.

Just as the *disable* action is the converse of the *enable* action, its data sharing behavior is intended to facilitate removal of data from the target pane's data dictionary rather than addition of data. When the *disable* action is performed on a pane (its `disable()` method called), the caller passes a list of all of its own data dictionary's keys. The target pane then can, if appropriate, use this list of keys to strip the caller's data (if any) out of its own data dictionary.

The data sharing interactions accompanying the *enable* and *disable* actions can be illustrated with the following example. Suppose that a pane named `button_pane` is to be enabled only if two other panes, named `entry_a` and

entry_b both have valid data. Initially the button_pane would be disabled, and then a sequence like the following might take place:

```
User enters a valid value into entry_a.
    entry_a automatically calls the enable() method of button_pane, passing its own␣
↪data.
    button_pane adds the data to its own dictionary.
    button_pane is still missing data from entry_b and so does not enable itself.
User enters a valid value into entry_b.
    entry_b automatically calls the enable() method of button_pane, passing its own␣
↪data.
    button_pane adds the data to its own dictionary.
    button_pane has data from both entry panes and so enables itself.
User edits entry_a, making the data invalid.
    entry_a automatically calls the disable() method of button_pane, passing its␣
↪dictionary keys.
    button_pane uses the keys to remove the corrresponding data from its own␣
↪dictionary.
    button_pane is now missing data from entry_a and so disables itself.
```

These interactions all happen automatically after the appropriate dependencies have been created between panes using the requires() method.

As this example illustrates, when one pane calls another pane's enable() method, the second pane will not necessarily actually be enabled if it does not have all the data it needs. Calling a pane's enable() method amounts to saying "Here's some data; use it to enable yourself if you can." The data that a pane needs to enable itself is established by calls to the requires() method.

TkPane objects also have a method named set_data that is intended as a standardized way to provide data to a pane. Similar to the enable method, it is intended to receive a data dictionary, but unlike the enable() method, it is not intended to call any callback lists. The information that is passed to a pane via the set_data() method may be used in any way: to update the pane's own data dictionary, to alter the pane's display (even adding or removing widgets), or trigger other actions. The operation of, and argument(s) to, the set_data() method are entirely specified by each pane subclass. None of the other TkPane class methods use the set_data() method, and so do not impose any requirements on it. Several of the pane classes in tkpane.lib recognize special keywords if they are present in a dictionary passed to set_data(). For example, the EntryPane class recognizes the key "prompt" as specifying a new value to be displayed on the pane as a prompt to the user; other keys and values are added to the EntryPane object's data dictionary.

1.3 Callback Lists

Inter-pane interactions are accomplished through lists of callback functions that each pane maintains, and uses to call the enable(), clear(), or disable() methods of other panes. Different callback lists are used for different conditions or state changes, such as data on the pane becoming valid or becoming invalid. Each callback list can contain callbacks to different action methods of several different panes.

The callback lists for each pane must be populated after the pane is instantiated. The most common callbacks are those associated with the *enable* and *disable* actions, and the callback lists used for these actions are automatically populated by the requires() method. There may be other types of inter-pane interactions that require these callback lists to be directly populated or modified during construction of the UI. Each pane has eight callback lists, corresponding to the following conditions:

- Data in a widget are changed and are valid (list on_change_data_valid).
- Data in a widget are changed and are invalid (list on_change_data_invalid).

- The user leaves the pane by changing the keyboard focus or moving the mouse, and the data are valid (list `on_exit_data_valid`).
- The user leaves the pane by changing the keyboard focus or moving the mouse, and the data are invalid (list `on_exit_data_invalid`).
- Data in the pane's data dictionary are changed (list `on_save_change`).
- The pane's `enable()` method is called (list `on_enable`).
- The pane's `clear()` method is called (list `on_clear`).
- The pane's `disable()` method is called (list `on_disable`).

All of the callback functions in each list are called whenever the corresponding condition occurs. For the *enable*, *clear*, and *disable* actions the callback functions are called after the pane itself is enabled, cleared, or disabled, respectively.

Some of these callback lists are automatically populated by the `requires()` and `can_use()` methods. All of the callback lists can be directly modified by appending appropriate callback function objects to them.

As described in the previous section, the `enable()` and `disable()` methods take different arguments: the first takes a dictionary, and the second takes a list of dictionary keys. The `clear()` method takes a list of dictionary keys just as does the `disable()` method. Any callback list may contain a mixture of `enable()` and `disable()` methods, and consequently different items in a callback list may require different arguments. The pane that calls those functions must know what argument to provide to each. To provide that information, the `tkpane` package provides several callback handler (`CbHandler`) classes that are used to associate appropriate information about argument types with each callback function. These `CbHandler` classes are documented below. The callback lists are lists of `CbHandler` objects, not simply lists of functions. If the callback lists must be directly addressed during UI construction, it is essential that callback functions be wrapped in `CbHandler` objects.

CHAPTER 2

Usage

The `tkpane` package defines a class, `TkPane`, that should be subclassed to create custom pane classes. The `TkPane` class itself does not include any Tkinter widgets, but only provides the interfaces and logic for simplifying inter-pane interactions.

Each custom pane class's constructor method (`__init__()`) must take at least one argument: the Tkinter widget, ordinarily a frame, that will be the parent for the pane itself. Custom pane classes may take additional arguments as needed, depending on the pane's functionality.

The general pattern for use of the `tkpane` package is:

1. Create any custom pane classes needed, or use those from `tkpane.lib`. See *Example 2* for an illustration of the creation of a custom pane class.
2. Create the layout of the application (or of an application window), defining a Tkinter frame or notebook page to hold each pane. See *Example 1* and *Example 3* for illustrations of creation of a layout that includes a frame for every pane that is to be used.
3. Instantiate a pane object for each enclosing frame by passing the enclosing frame widget to the pane class's constructor method.
4. If needed, use the `requires()` method to identify which frames depend on valid data from other frames. See, for example, lines 57 and 58 of *Example 1*, and lines 116 and 117 of *Example 3*.
5. If needed, use the `requires_datavalue()` method to indicate that the pane must have a specific key and value in its data dictionary for it to enable itself.
6. If needed, use the `can_use()` method to indicate that one pane will accept another pane's data, but does not require it.
7. If needed, append any additional required callback functions, encapsulated in callback handler objects, to the pane's *callback lists*. See, for example, lines 121 and 122 of *Example 3*.
8. If desired, set the status and progress reporting objects for individual panes, assigning appropriate pane objects to the `status_reporter` and `progress_reporter` attributes of the `TkPane` subclass object. See, for example, lines 66 and 67 of *Example 1*.
9. If needed, perform any additional pane-specific or application-specific configuration or initialization. See, for example, line 71 of *Example 1* and lines 126 and 127 of *Example 3*.

10. If needed, initialize the appearance and data sharing of the UI by calling the `tkpane.run_validity_callbacks()` and `tkpane.enable_or_disable_all()` functions, passing each a list of panes as an argument. See, for example, line 62 of *Example 1*.

CHAPTER 3

Interface

Objects of `TkPane` subclasses have numerous attributes and methods that can be modified or called directly, but there is a set of methods that is specifically intended to be used as an interface to TkPane objects. These interface methods provide the functionality needed for inter-pane interactions, and also may be used by other application code. The interface methods are:

all_data() Returns the pane's entire data dictionary. This contains both the data values managed by the pane itself as well as any other data values that the pane has been sent by, or requested from, other panes. A custom subclass might use other internal data storage mechanisms (e.g., class attributes), and if so, those data values will not appear in this data dictionary. If no valid data values have been entered into the pane, they (and their keys) will not be present in the data dictionary that is returned.

can_use(other_pane) Takes another pane as an argument and sets up callbacks so that a) the other pane will send to this pane either data (if the other pane's data are valid) or a list of invalid data keys (if the other pane's data are invalid); and b) if the other pane's data are cleared, this pane's data will be cleared also.

clear(key_list) Takes as an argument a list of keys to identify data items that might be maintained in the pane's internal data dictionary. The corresponding data values, as well as all data managed by the pane itself, will be removed from the pane's data dictionary. Any data displayed on the pane's widgets will also be removed (subject to the subclass' implementation of the `clear_pane()` method). The callbacks in the pane's `on_clear` callback list will all be called.

clear_data(key_list) Takes as an argument a list of keys to identify data items that might be maintained in the pane's dictionary. The corresponding data values will be removed from the pane's data dictionary. If the given list of keys includes *any* of the pane's own data keys, then this method will behave the same as the `clear()` method.

clear_on_disable() Causes a pane to clear itself when it is disabled. Note that this is different from the "clear_on_disable" argument to the `requires()` method: the latter clears the pane when the *other* pane is disabled, whereas this method clears the pane when this pane is itself disabled.

disable(key_list) Takes as an argument a list of keys to identify data items that might be maintained in the pane's dictionary. The corresponding data values will be removed from the pane's data dictionary and widgets on the pane will be disabled (subject to the subclass' implementation of the `disable_pane()` method). The callbacks in the pane's `on_disable` callback list will all be called.

enable(data_dictionary) Takes as an argument a dictionary containing data values that the pane may make use of. If, after receipt of these data, the pane has all of the data values that it needs to enable itself, it will enable itself.

This method is used by TkPane objects as part of the management of inter-pane interactions, but may also be useful to call directly.

focus() Sets the focus to an appropriate widget on this pane. This method must be overridden by subclasses that implement panes that contain widgets that can take the focus.

requires(other_pane) Takes another pane as an argument and sets up callbacks so that a) changes in the validity of data on the other pane will cause this pane to become enabled or disabled, and will send to this pane either data (if the other pane's data are valid) or a list of invalid data keys (if the other pane's data are invalid); and b) if the other pane's data are cleared, this pane's data will be cleared also. This also marks the other pane's data as required.

requires_datavalue(key, value) Takes a data key and a data value, or list of values, and ensures that the pane will only be enabled when the pane's data dictionary contains the specified data value (or any of the values, if more than one is specified) for the given key. This method may be called multiple times using the same key; the data values provided will be compiled into a list. When the pane's data dictionary value for the given key is a list (as it will be if the data was provided by a multiple-select listbox), then the pane will be enabled if any of the items in that list match the specified data value (or any of the values in the list, if the specified data value is a list). Commonly the key and value used as arguments to this method will be obtained from another pane (e.g., via the `enable()` method), but this method does not refer explicitly to any other pane. Often when the `requires_datavalue()` method is used, it will be appropriate to also use the `requires()` method with an argument identifying the other pane that provides the data key and value. This method is useful when one pane (e.g., one containing radio buttons) must enable different other panes depending on its own value.

set_data(data_dictionary) A generalized method intended to be used to send data or other information to a pane. It can be used, for example, to send a dictionary of data to a pane, similar to the `enable()` method, but without enabling the pane if it is disabled. The actual behavior of this method, however, and the arguments that it takes, can be overridden in each subclass.

set_key(key_name) [or set_keys(key_name1, key_name2, . . .)] Changes the key (or keys) used to identify data items managed by this pane in its internal data dictionary.

valid_data() Returns a Boolean indicating whether or not valid data have been entered into the pane. This method must be overridden by subclasses that manage data.

values() Returns a dictionary of just the data values managed by the pane. If no valid data values have been entered into the pane, they will not appear in the data dictionary that is returned.

CHAPTER 4

Creating Custom Panes by Subclassing TkPane

The pane library (`tkpane.lib`) contains several general-purpose custom panes that may be used directly or used as templates for creation of other custom panes. These panes may be all that is needed to build some applications, but the `TkPane` class is meant to be subclassed so that more application-specific panes can be created as needed.

There are three general types of panes that might be created by subclassing the TkPane class, and the `tkpane.lib` package contains custom subclasses that illustrate each of these, and that can be used as templates for other custom subclasses:

- Panes that handle user-entered data, where the data may be invalid (e.g., when required and missing). See `tkpane.lib.EntryPane` and *Example 2*.
- Panes that handle user-entered data where the data are constrained to never be invalid. See `tkpane.lib.ScalePane` and `tkpane.lib.UserPane`.
- Panes that do not handle user-entered data. See `tkpane.lib.MessagePane` and `tkpane.lib.ButtonPane`.

Each custom pane must call the TkPane class's own initialization method, passing it the following values:

- The Tkinter parent widget.
- The name to be used for this pane (primarily for status reporting).
- A dictionary of Tkinter frame configuration options to be applied to the TkPane's own enclosing frame.
- A dictionary of Tkinter gridding options to be applied to the TkPane's frame.

This initialization line might look like:

```
tkpane.TkPane.__init__(self, parent, pane_name, tkpane.lib.frame_config_opts(),␣
↪tkpane.lib.frame_grid_opts())
```

Each custom pane that manages data should also initialize the `datakeylist` attribute. This should be a list of the dictionary key names for the data values that are managed by the pane.

Each custom pane that manages data should also override the following six methods of the TkPane class:

entry_widgets() Return a list of the Tkinter widgets that are used for data entry.

valid_data(entry_widget=None) Return a Boolean indicating whether data in the specified widget are valid. If no widget is specified, all entry widgets in the pane should be evaluated.

save_data(is_valid, entry_widget=None) Modify the pane's data dictionary, either updating it from the widget if the widget's data are valid, or clearing the relevant data if the widget's data are not valid.

clear_pane() Remove all data displayed in the pane's widgets.

enable_pane() Enable all widgets on the pane.

disable_pane() Disable all widgets on the pane.

CHAPTER 5

Validating Data

Each custom TkPane subclass that handles data must define a data validation method named `valid_data`. This method will be called on every change to data in the pane's widget(s). For example, for the `EntryPane` class in `tkpane.lib`, the validation method will be called after every character is typed. This method must return True or False to indicate whether or not the pane's data are valid. A return value of False does not cause the change just made to be rejected or rolled back, but the validation method of a pane may directly manipulate the pane's widgets or Tkinter variables, or call other pane methods such as `set_data()` to alter the data.

Pane classes defined in `tkpane.lib` generally consider a pane's data to be invalid if a data value is required but missing. Several of the library pane classes apply additional criteria; for example, the InputFilePane class requires that the specified file exist, and the OutputDirPane and OutputFilePane classes both require that directories exist.

Several of the pane classes in `tkpane.lib` also allow custom validation functions to be applied. These classes each have a method that allows a data validation callback function to be identified. These validation functions should accept data value(s) from the pane and return True or False indicating the validity of the data. The methods used to specify data validation callbacks for different library pane classes are:

- EntryPane: set_entry_validator()
- InputFilePane: set_filename_validator()
- InputFilePane2: set_filename_validator()
- TextPane: set_entry_validator()
- UserPane: set_user_validator()
- UserPasswordPane: set_user_validator()

CHAPTER 6

Setting TkPane Frame Styles

When a custom pane class's initialization method calls the `TkPane` class's initialization method, it must provide dictionaries of frame configuration and gridding options The `tkpane.lib` package provides a number of pre-defined pane styles, and the ability to create new styles, to simplify this process and easily standardize pane appearance. The pane style definitions have no correspondence with Style classes that can be used with ttk widgets.

This is carried out by instantiating a `PaneStyle` object, which is initialized with a style name, a dictionary of frame configuration options, and a dictionary of frame configuration options. The option dictionaries for any named style can then be obtained using the following methods of `tkpane.lib`:

frame_config_opts(style_name) Return the dictionary of frame configuration options for the specified style.

frame_grid_opts(style_name) Return the dictionary of frame gridding options for the specified style.

The `tkpane.lib` variable `current_panestyle` can be set to a style name to ensure that all custom panes from `tkpane.lib` that are subsequently instantiated use that style.

Widget and pane spacing in `tkpane.lib` follow the GNOME guidelines for visual layout. As a rule, each widget is surrounded by 3 pixes of spacing in both X and Y directions, so adjacent widgets on the same pane are separated by 6 pixels. Several pre-defined pane styles are provided in `tkpane.lib` that are generally consistent with these guidelines but provide some variety in appearance. These specify different frame configuration options; none of them specify any frame gridding options. The provided pane styles are:

default Six pixels of internal padding within the enclosing frame in both the X and Y directions. In combination with the three-pixel spacing around widgets, this results in widgets on adjacent panes being separated by 18 pixels. As the name implies, this is the default setting for `tkpane.lib`.

plain No internal frame padding in either the X or Y directions.

closex Six pixels of internal frame padding in the Y direction and no padding in the X direction. This is appropriate for a series of horizontally-adjacent ButtonPane panes.

closey Zero pixels of padding in the Y direction and six pixels of padding in the Y direction. This is appropriate for a series of vertically-adjacent panes containing individual checkboxes or radio buttons.

ridged Six pixels of padding in both the X and Y directions and a two-pixel border width with a ridged border style.

grooved Six pixels of padding in both the X and Y directions and a two-pixel border width with a grooved border style.

sunken Six pixels of padding in both the X and Y directions and a two-pixel border width with a sunken border style.

statusbar Six pixels of padding in the X direction, two pixels of padding in the Y direction, and a two-pixel border width with a sunken border style. As the name implies, this is appropriate for status bars and is used by the StatusProgressPane in `tkpane.lib`.

CHAPTER 7

Highlighting Invalid Data

By default, if a pane's data is designated as required (e.g., by reference to the pane in the `requires()` method of another pane), and the pane's data are invalid, the widget's background is colored a light red, as shown in the following figure.

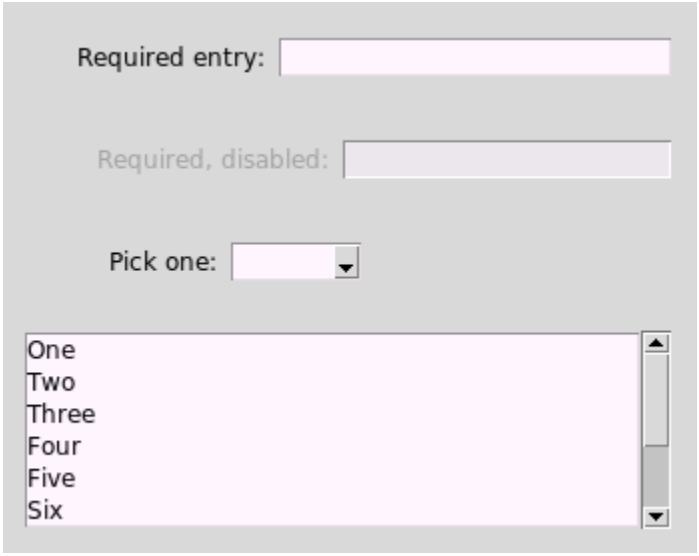

Fig. 1: Figure 2. Coloring of Widgets with Invalid Data

The default colors used to indicate that an entry is required but invalid are module-level variables:

`invalid_color` The color used for the widget background when the data are required but invalid, and the widget is not disabled.

`invalid_disabled_color` The color used for the widget background when the data are required but invalid, and the widget is disabled. This is only applied to ttk widgets.

In addition to differences in the handling of the `invalid_disabled_color` for Tkinter and ttk widgets, there are differences between Linux and Windows: when using the default theme on Windows, background colors of ttk

widgets are unaffected.

These default colors are picked up by each TkPane subclass when it is instantiated. If you change the module-level variables, the new colors will be used by all subsequently created widgets. The 'invalid' colors for an already-created pane can be changed with the `set_invalid_color()` method of the TkPane class.

The `invalid_color` setting also serves as a switch to disable changes to the background color of widgets with required but invalid data. Setting `invalid_color` to None will turn off the automatic changes to widget background colors.

Note that some widgets (including the checkbox and scale) do not respond to any changes in background colors. These two widgets, however, should not have any invalid state.

CHAPTER 8

Changing Widget Types in `tkpane.lib`

The pane classes in `tkpane.lib` use ttk widgets by default (if a ttk version of a widget exists). Tkinter widgets can be used instead, as determined by the `tkpane.use_ttk` configuration variable. If this is set to False, subsequent panes instantiated from `tkpane.lib` will use Tkinter widgets instead of ttk widgets, where possible.

CHAPTER 9

Package Contents

Creating a Tkinter layout is done with the `TkPane` class in the `tkpane` module.

9.1 The TkPane Class

class tkpane.**TkPane**(*parent*, *pane_name*, *config_opts=None*, *grid_opts=None*)

Base class for Tkinter UI elements (panes). This class is meant to be subclassed,

Subclasses are expected to call the TkPane class' initialization function.

Parameters

- **parent** – The parent widget for this pane (ordinarily a frame or top-level element).
- **config_opts** – A dictionary of keyword arguments for configuring the pane's frame.
- **grid_opts** – A dictionary of keyword arguments for gridding the pane's frame.

Attributes of a TkPane object that may be assigned after instantiation are:

- required: A Boolean indicating whether valid data must be entered. Note that this applies to all widgets on the pane. If some data values are required by the application and others are not, the widgets for those different data values should be on different panes.
- datakeylist: A list of dictionary keys for data items managed by this pane.
- datadict: A dictionary containing all data managed or used by this pane.
- on_change_data_valid: A list of CbHandler object to be called when data are changed and valid.
- on_change_data_invalid: A list of CbHandler objects to be called when data are changed and invalid.
- on_exit_data_valid: A list of CbHandler objects to be called when the pane loses focus and the data are valid.
- on_exit_data_invalid: A list of CbHandler objects to be called when the pane loses focus and the data are invalid.

- on_save_change: A list of CbHandler objects to be called when the values of the pane's own data keys are modified in the pane's internal data dictionary.
- on_enable: A list of CbHandler objects to be called when this pane is enabled.
- on_clear: A list of CbHandler objects to be called when this pane is cleared.
- on_disable: A list of CbHandler objects to be called when this pane is disabled.
- keys_to_enable: Keys of datadict that must be defined for this pane to be enabled.
- data_to_enable: A dictionary of data values that are required for this pane to be enabled. The keys of this dictionary are keys of self.datadict and the value for each key is a list of allowable values for that key.
- status_reporter: An object with a well-known method (set_status) for reporting status information.
- progress_reporter: An object with well-known methods (set_determinate, set_indeterminate, set_value, start, stop) for reporting or displaying progress information.

Methods to be overridden by subclasses that manage data are:

- save_data(is_valid): Updates or clears the data dictionary with widget values, depending on whether the data are valid.
- valid_data(widget=None): Evaluates whether data entered into widgets on the pane are valid, Returns True or False.
- entry_widgets(): Returns a list of widgets used for data entry.
- enable_pane(): Enable widgets on this pane.
- clear_pane(): Clear data from widgets on this pane.
- disable_pane(): Disable widgets on this pane.

Other methods that may commonly be overridden by subclasses are:

- focus
- set_data

Subclasses must create all the widgets on the pane and configure their appearance and actions. Interactions between panes should be managed using the lists of CbHandler callbacks.

all_data()
Return all data in this pane's data dictionary, as a dictionary.

can_enable()
Determine whether all the required data values are available for this pane to actually enable itself.

Returns True or False.

can_use(*other_pane*)
Set handler functions so that the other pane can provide data for the data dictionary, or keys of data to remove.

Parameters **other_pane** – The pane which must have valid data for this pane to be enabled.

The other pane's lists of Handler callbacks are modified to add the other pane's data to this pane's data dictionary or to clear the other pane's data keys from this pane's data dictionary when the other pane's data are valid or invalid, respectively.

clear(*keylist=[]*)
Re-initialize any data entry or display elements on the pane, and remove stored data.

Parameters **keylist** – A list of keys for data to be removed from the pane's data dictionary.

This method will remove data for the given keys from the pane's data dictionary, *and also* remove all of the pane's own data. Any data displayed on the pane will be cleared (subject to the subclass' implementation of the *clear_pane()*' method). All of the `on_clear` callbacks will be called.

clear_data(*keylist=None*)
Remove the values for the specified keys from the pane's data dictionary.

Parameters **data_keys** – A list of keys of the dictionary entries to remove.

This method removes some or all data values not managed by this pane from the pane's data dictionary.

If any of the specified keys are for the data managed by this pane, then *all* of the data managed by this pane will be cleared, as well as any other values in keylist; the pane will be cleared; and all callbacks in the `on_clear` list will be called–that is, this method will act as if the `clear()` method had been called instead.

If no data keys are specified, all data values that are *not* managed by this pane will be removed.

clear_on_disable()
Clears the pane's own data and pane (widgets) when the pane is disabled.

clear_own()
Clears the pane's own data from its data dictionary.

This is intended primarily for use by subclasses. It does not clear the pane's visible data, so it can lead to inconsistencies if not used properly.

clear_pane()
Clear data from widgets on this pane, as appropriate.

This method should be overridden by child classes. This method is not meant to be called directly, but only called indirectly via the clear() method.

disable(*keylist=[]*)
Disable the pane so that the user can't interact with it.

Parameters **keylist** – A list of the keys to be removed from the pane's data dictionary.

This method may be overridden by child classes if simply overriding the disable_pane() method is not sufficient to implement the needed behavior.

disable_pane()
Disable any widgets on this pane that are necessary for user interaction.

This method should be overridden by child classes. This method is not meant to be called directly, but only called indirectly via the disable() method.

enable(*incoming_data={}*)
Enable this pane (subject to data requirements being met).

Parameters **incoming_data** – A dictionary of data from the caller.

The incoming data will be merged into this pane's own data dictionary.

enable_pane()
Enable any widgets on this pane that are necessary for initial user interaction.

This method should be overridden by child classes. This method is not meant to be called directly, but only called indirectly via the enable() method.

entering(*event*)
Record the initial data value to be used later to determine if it has changed.

entry_widgets()
Return a list of entry widgets on the pane.

The purpose of this method is for the TkPane to automatically recolor all entry widgets based on their validity. Defaults to returning an empty list.

This method should be overridden by any subclass that manages data.

focus()
Set the focus of this pane to the appropriate widget.

This method should be overridden by subclasses. It is part of the standard interface of a pane, and has no action in the base TkPane class.

handle_change_validity(*is_valid*, *entry_widget=None*)
Update the data dictionary from pane widgets and call appropriate handlers for data changes.

Parameters

- **is_valid** – A Boolean indicating whether or not data on the pane are valid.
- **entry_widget** – The widget that has been changed. Its state will be changed as appropriate to indicate the data validity.

If entry_widget is not provided, all widgets will have their state changed to indicate the data validity.

handle_exit_validity(*is_valid*, *widget_list=None*)
Update the data dictionary from pane widgets and call appropriate handlers for pane exiting.

Parameters

- **is_valid** – A Boolean indicating whether or not data on the pane are valid.
- **widget_list** – A list of widgets to which 'is_valid' applies. Their states will be changed as appropriate to indicate the data validity.

If widget_list is not provided, all widgets will have their state changed to indicate the data validity.

leaving(*event*)
Revise the data dictionary, call all exit handlers, and report status.

Revision of the data dictionary is carried out through the save_data method, which should be overridden by subclasses. The appropriate set of exit handlers is called depending on sata validity.

report_progress(*progress_value*)
Send the given progress value to the progress reporting function, if it is defined.

report_status(*status_msg*)
Send the given message to the status reporting function, if it is defined.

requires(*other_pane*, *enable_on_other_exit_only=False*, *disable_on_other_exit_only=False*, *clear_on_enable=False*, *clear_on_disable=False*)
Set handler functions for the other pane to enable or disable this pane.

Parameters

- **other_pane** – The pane which must have valid data for this pane to be enabled.
- **enable_on_other_exit_only** – A Boolean indicating whether this pane should be enabled only when the other pane exits with valid data (True) or any time the other pane's data become valid (False). Default: False.
- **disable_on_other_exit_only** – A Boolean indicating whether this pane should be disabled only when the other pane exits with invalid data (True) or any time the other pane's data become invalid (False). Default: False.

- **clear_on_enable** – A Boolean indicating whether this pane should be cleared when it is enabled. Default: False.
- **clear_on_disable** – A Boolean indicating whether this pane should be cleared when it is disabled. Default: False.

The other pane's lists of CbHandler callbacks are modified to enable or disable this pane when the other pane's data are valid or invalid, respectively. This pane's 'clear-data()' method is also added to the other pane's 'on_clear' callback list. The other pane's 'required' attribute is also set to True. Optionally, this pane can also be cleared when it is either enabled or disabled as a result of data validity changes in the other pane.

requires_datavalue(*key*, *value*)
Specify that a particular data value, for a specific data key, is required for the pane to enable itself.

Parameters

- **key** – The key for the data value, that must be in the pane's data dictionary.
- **value** – The value, or list of values, that must be in the pane's data dictionary for that key.

This method does not create any callbacks that cause the other pane that provides the data value to enable or disable the current pane. When the `requires_datavalue()` method is used, it may be appropriate to also use the `requires()` method, specifying the other pane that provides those data values.

save_data(*is_valid*, *entry_widget=None*)
Update the pane's data dictionary with data from entry widgets.

This may add a value, revise a value, or remove a key (if not is_valid). This may also change widget displays, e.g., setting invalid values to empty strings or the equivalent.

This method ordinarily should be overridden by any subclass that manages data.

send_status_message(*is_valid*)
Send a status message reporting data values and/or validity if data have changed.

Parameters **is_valid** – A Boolean indicating whether or not the data on the pane are valid.

This method may be overridden.

set_data(*data*)
Accepts data to be used by the pane as appropriate.

Parameters **data** – This is intended to be a dictionary, but subclasses that override this method can accept any number of parameters of any type.

This method allows other panes or application code to send data to a pane without triggering any callbacks. The use of this data and even the type of data may be redefined by the custom pane subclass, and should be respected by the caller. No other TkPane methods use this method; it exists only to provide a uniform data-passing method for all subclasses.

set_invalid_color(*invalid_color*, *invalid_disabled_color*)
Save the colors to be applied to any invalid widgets, and immediately apply them.

set_style(*ttk_style*)
Set the style of the widget(s) on this pane.

This method should be overridden by subclasses. It is part of the standard interface of a pane, and has no action in the base TkPane class.

show_widget_validity(*widget*, *is_valid*)
Set the widget's background color to the 'valid' or 'invalid' color.

tk and ttk widgets are handled differently. This implementation assumes that Text, Listbox, and Spinbox widgets are all tk widgets, and all others are ttk widgets (as in tkpane.lib). If other types of widgets are used (e.g., a tk.Entry widget), this method will have to be overridden.

show_widgets_validity()
Set all widgets' background color to the 'valid' or 'invalid' color.

valid_data(*entry_widget=None*)
Evaluate whether data entered into one or all widgets on the pane are valid,

Returns True or False. Defaults to returning True. This method must be overridden by any subclass that manages data.

values()
Return data values managed by this pane as a dictionary.

If the 'datakeylist' and 'datadict' attributes are managed properly, this function will work as intended, but subclasses may need to override this method if they are not using the datadict to store the pane's data.

9.2 Action Method Handling Classes (CbHandler Classes)

Every callback function used by a pane must be wrapped in one of the following callback handler (CbHandler) classes. This conversion of callback functions to CbHandler objects is performed automatically by the `requires()` method of the `TkPane` class. If items are to be added to the callback lists directly by the UI implementer, these CbHandler classes must be used.

class tkpane.**CbHandler**(*function*)
Define a callback function that takes no arguments.

class tkpane.**PaneDataHandler**(*function*)
Define a callback function that takes a single argument consisting of a dictionary of a pane's own data values.

class tkpane.**PaneKeyHandler**(*function*)
Define a callback function that takes a single argument consisting of a list of a pane's own data keys.

class tkpane.**AllDataHandler**(*function*)
Define a callback function that takes a single argument consisting of a dictionary of all of a pane's data values.

9.3 Functions

Several functions in the `tkpane` package simplify the initialization of a UI or the integration of the `tkpane` and `tklayout` libraries.

tkpane.**run_validity_callbacks**(*panelist*)
Run the 'on_exit_data_(in)valid' callbacks for all panes in the list,

This function is intended to be used after all panes have been instantiated, when some panes have default values and are required by other panes. This function will ensure that the dependent panes have the required initial data values.

tkpane.**enable_or_disable_all**(*panelist*)
Enable or disable all panes in the list, as required by their data status.

This function is intended primarily to be used after the UI has been built but before it has been activated, to ensure that all of the specified panes are initallly appropriately enabled or disabled.

tkpane.**layout_panes**(*layout*)

Take a tklayout.AppLayout object and return a dictionary of pane objects with AppLayout identifiers as keys,

If the `tklayout` package has been used to assemble panes into a UI, this function will create and return a dictionary of all of the AppLayout element names and corresponding pane objects. This can simplify access to the pane objects for subsequent customization.

tkpane.**build_ui**(*layout_spec*, *tk_parent*, *ui_root_element*, *build_specs*)

Take specifications for a UI layout and panes, and assemble them, returning a dictionary of panes.

Parameters

- **layout_spec** – A tklayout.AppLayout object for which all of the UI elements have been described.
- **tk_parent** – The Tkinter widget (e.g., a frame) that will be the parent widget for the entire UI.
- **ui_root_element** – The name of the layout element (in 'layout_spec') that is the container for all other layout elements.
- **build_specs** – A dictionary of layout element names and functions to populate those elements with panes (as is needed by tklayout.build_elements).

This is a convenience function that integrates operations of the `tklayout` and `tkpane` packages to minimize application code.

9.4 The Pane Library

The pane library (`tkpane.lib`) contains a number of custom, but general-purpose, subclasses of the `TkPane` class. These pane classes may be used directly in an application, or may be used as templates for the creation of other pane classes.

class tkpane.lib.**ButtonPane**(*parent*, *button_text*, *pane_name='button'*, *button_action=None*, *width=None*)

Display a simple text button with configurable text and action.

Parameters

- **pane_name** – The name to be used to identify this pane in status messages.
- **button_text** – The text to display on the button.
- **button_action** – A callback to perform an action when the button is clicked.
- **width** – The width of the button, in characters (optional).

There are no data keys specific to this pane.

Overridden methods:

- disable_pane
- enable_pane
- set_style
- focus
- set_data

Custom methods:

- set_button_action

- do_button_action

do_button_action()
Trigger this pane's action. The callback function will be passed this pane's data dictionary.

focus()
Set the focus to the button.

set_button_action(*button_action*)
Specify the callback function to be called when the button is clicked.

set_data(*data_dict*)
Update the pane's data dictionary with the provided data.

Special keys are: * button_text: Contains the text to place on the button. * button_width: Contains the width for the button.

All other data in the passed dictionary are added to the button's own data dictionary.

class tkpane.lib.**CanvasPane**(*parent*, *width=None*, *height=None*, *config_opts=None*)
Display a Tkinter Canvas widget.

Parameters

- **width** – The width of the Canvas widget, in pixels (optional).
- **height** – The height of the Canvas widget, in pixels (optional).
- **config_opts** – A dictionary of configuration options for the Canvas widget (optional).

Because of the variety of types of information that can be displayed on a Canvas widget, and associated metadata (e.g., position), the CanvasPane class does not maintain a data dictionary representing any of that information. Nor is there any built-in determination of whether the Canvas' contents are valid or invalid. The `canvas_widget()` method should be used for direct access to the Canvas widget, to either add or access data.

Overridden methods:

- clear_pane
- enable_pane
- disable_pane
- focus

Custom methods:

- canvas_widget
- set_scrolling

canvas_widget()
Return the canvas widget object, to allow direct manipulation.

clear_pane()
Clear the canvas.

disable_pane()
Disable the canvas.

enable_pane()
Enable the canvas.

focus()
Set the focus to the canvas widget.

set_scrolling()
Set or reset the scrollbar limits to allow scrolling of the entire contents of the Canvas.

class tkpane.lib.**CheckboxPane**(*parent*, *pane_name*, *prompt*, *valid_state=None*, *key_name=None*, *config_opts=None*)
Display a Tkinter Checkbutton widget to accept True and False values.

Parameters

- **pane_name** – The name to be used to identify this pane in status messages.
- **prompt** – The text associated with the checkbox.
- **valid_state** – Either True or False to indicate that either the checked or unchecked state (only) is to be considered valid. If not specified (the default), the checkbox will always be considered to have valid data.
- **key_name** – The name to be used with the internal data dictionary to identify the entry data; use to avoid name conflicts with other CheckboxPane panes on the same UI (optional).
- **config_opts** – A dictionary of configuration options for the Checkbutton widget.

Data keys managed by this pane: "check" or the key name specified during initialization.

The value of this item is always True or False, and defaults to False.

Name used by this pane: user-defined on initialization.

Overridden methods:

- entry_widgets
- valid_data
- save_data
- clear_pane
- enable_pane
- disable_pane
- set_style
- focus
- set_data

Custom methods:

- set_key

focus()
Set the focus to the checkbox.

save_data(*is_valid*, *entry_widget*)
Update the pane's data dictionary with data from the Checkbutton widget.

set_data(*data*)
Update the pane's data dictionary with the provided data.

Special key supported: "prompt" changes the pane's prompt.

set_key(*key_name*)
Change the name of the data key used for the entered data.

Parameters **key_name** – New name for the data key.

This method allows the name of the data key to be customized to eliminate conflicts with other CheckboxPane objects on the same UI.

valid_data(*widget=None*)
Returns an indication of whether the checkbox is in a valid state.

class tkpane.lib.**ComboboxPane**(*parent*, *pane_name*, *prompt*, *items*, *item_only=False*, *key_name=None*)
Display a Tkinter Combobox widget with a prompt.

Parameters

- **pane_name** – The name to be used to identify this pane in status messages.
- **prompt** – The prompt to be presented in a Label widget adjacent to the entry.
- **items** – The list of items to be included in the drop-down list.
- **item_only** – A Boolean indicating whether or not items from the list are the only valid entries. Default: False.
- **key_name** – The name to be used with the internal data dictionary to identify the entry data; use to avoid name conflicts with other EntryPane objects on the same UI (optional).

Data keys managed by this pane: "combobox" or the key name specified during initialization.

Name used by this pane: user-defined on initialization.

Overridden methods:

- entry_widgets
- valid_data
- save_data
- clear_pane
- enable_pane
- disable_pane
- set_style
- focus
- set_data

Custom method:

- set_key

focus()
Set the focus to the entry.

save_data(*is_valid*, *entry_widget*)
Update the pane's data dictionary with data from the Combobox widget.

set_data(*data*)
Update the pane's data dictionary with the provided data.

Special key supported: 'prompt' changes the pane's prompt.

set_key(*key_name*)
Change the name of the data key used for the entered data.

Parameters **key_name** – New name for the data key.

This method allows the name of the data key to be customized to eliminate conflicts with other EntryPane objects on the same UI.

set_newitems(*items*)
Change the items displayed in the list. This will clear any selection.

class tkpane.lib.**EmptyPane**(*parent*)
A pane with no widgets that can be used as a spacer.

class tkpane.lib.**EntryPane**(*parent*, *pane_name*, *prompt*, *key_name=None*, *required=False*, *blank_is_valid=False*)
Display a Tkinter Entry widget.

Parameters

- **pane_name** – The name to be used to identify this pane in status messages.
- **prompt** – The prompt to be presented in a Label widget adjacent to the entry.
- **key_name** – The name to be used with the internal data dictionary to identify the entry data; use to avoid name conflicts with other EntryPane objects on the same UI (optional).
- **required** – A Boolean indicating whether valid data must be entered (optional; default is False).
- **blank_is_valid** – A Boolean indicating whether, if an entry is not required, a blank value should be treated as a valid entry (optional; default is False). If this is set to True, an empty string may be passed to any other pane that requires this pane.

Data keys managed by this pane: "entry" or the key name specified during initialization.

Name used by this pane: user-defined on initialization.

Overridden methods:

- entry_widgets
- valid_data
- save_data
- clear_pane
- enable_pane
- disable_pane
- set_style
- focus
- set_data

Custom method:

- set_key
- set_entry_validator

focus()
Set the focus to the entry.

save_data(*is_valid*, *entry_widget*)
Update the pane's data dictionary with data from the Entry widget.

set_data(*data*)
Update the pane's data dictionary with the provided data.

Special key supported: 'prompt' changes the pane's prompt.

set_entry_validator(*fn*)
Set the callback function that will be used to check the entered value.

This function must take the entry value as an argument and return a Boolean.

set_key(*key_name*)
Change the name of the data key used for the entered data.

Parameters **key_name** – New name for the data key.

This method allows the name of the data key to be customized to eliminate conflicts with other EntryPane objects on the same UI.

class tkpane.lib.**InputFilePane**(*parent*, *optiondict=None*)
Get and display an input filename.

Parameters **optiondict** – a dictionary of option names and values for the Tkinter 'askopenfilename' method (optional).

Data key managed by this pane: "input_filename".

Name used by this pane: "Input filename".

Overridden methods:

- entry_widgets
- valid_data
- save_data
- clear_pane
- disable_pane
- enable_pane
- set_style
- focus
- set_data

Custom methods:

- set_key
- set_filename_validator

entry_widgets()
Return a list of widgets used for data entry.

focus()
Set the focus to the entry.

save_data(*is_valid*, *entry_widget*)
Update the pane's data dictionary with data from the entry widget.

Overrides TkPane class method.

set_data(*data*)
Update the pane's data dictionary with the provide data.

Special key supported: 'input_filename' changes the filename in the entry widget.

set_filename_validator(*fn*)
Set the callback function that will be used to check the entered filename.

This function must take the filename as an argument and return a Boolean.

set_key(*key_name*)
Change the name of the data key used for the entered data.

Parameters **key_name** – New name for the data key.

This method allows the name of the data key to be customized to eliminate conflicts with other InputFilePane objects on the same UI.

valid_data(*widget=None*)
Return True or False indicating the validity of the filename entry.

Overrides TkPane class method.

class tkpane.lib.**InputFilePane2**(*parent*, *prompt='Input file:'*, *optiondict=None*, *key_name=None*)
Get and display an input filename. This class accepts a prompt and uses a different widget layout than InputFilePane.

Parameters

- **prompt** – A prompt presented above the text box for entry of the filename (optional; default="Input file:").
- **optiondict** – A dictionary of option names and values for the Tkinter 'askopenfilename' method (optional).

Key_name A name to use as a key for the data value (filename) managed by this pane (optional default="input_filename").

Data key managed by this pane: "input_filename" or the key name specified during initialization.

Name used by this pane: "Input filename".

Overridden methods:

- entry_widgets
- valid_data
- save_data
- clear_pane
- disable_pane
- enable_pane
- set_style
- focus
- set_data

Custom methods:

- set_key
- set_filename_validator

entry_widgets()
Return a list of widgets used for data entry.

focus()
Set the focus to the entry.

save_data(*is_valid*, *entry_widget*)
Update the pane's data dictionary with data from the entry widget.

Overrides TkPane class method.

set_data(*data*)
Update the pane's data dictionary with the provided data.

Special key supported: 'input_filename' changes the filename in the entry widget.

set_filename_validator(*fn*)
Set the callback function that will be used to check the entered filename.

This function must take the filename as an argument and return a Boolean.

set_key(*key_name*)
Change the name of the data key used for the entered data.

Parameters **key_name** – New name for the data key.

This method allows the name of the data key to be customized to eliminate conflicts with other InputFilePane objects on the same UI.

valid_data(*widget=None*)
Return True or False indicating the validity of the filename entry.

Overrides TkPane class method.

class tkpane.lib.**ListboxPane**(*parent*, *pane_name*, *items*, *rows=None*, *width=None*, *key_name=None*, *mode=None*)
Display a Tkinter Listbox.

Parameters

- **pane_name** – The name to be used to identify this pane in status messages.
- **items** – The list of items to be initially displayed in the listbox.
- **rows** – The number of rows (items) to be shown; the listbox will have a scrollbar (optional).
- **key_name** – The name to be used with the internal data dictionary to identify the selected list entries; use to avoid name conflicts with other ListboxPane objects on the same UI (optional).
- **mode** – The selection mode to use; "single", "browse", "multiple", or "extended" (optional; default is "extended").

Data key managed by this pane: "listbox" or the key name specified during initialization.

The value of the data managed by this pane is a list of the selected items.

Name used by this pane: user-defined on initialization.

Overridden methods:

- entry_widgets
- valid_data
- save_data
- clear_pane
- enable_pane

- disable_pane
- set_style
- focus

Custom methods:

- set_newitems
- set_key

focus()
Set the focus to the listbox.

save_data(*is_valid*, *entry_widget*)
Update the pane's data dictionary with data from the Listbox widget.

set_key(*key_name*)
Change the name of the data key used for the entered data.

Parameters **key_name** – New name for the data key.

This method allows the name of the data key to be customized to eliminate conflicts with other ListboxPane objects on the same UI.

set_newitems(*items*)
Change the items displayed in the list. This will clear any selection.

set_style(*ttk_style*)
Sets the style of the scrollbar accompanying the listbox.

The Listbox widget is not a themed ttk widget and cannot have a style applied.

valid_data(*widget=None*)
Returns an indication of whether the listbox is both required and has at least one selection.

class tkpane.lib.**MessagePane**(*parent*, *message*)
Display a text message.

Parameters **message** – The message to display..

This pane does not manage any data.

Name used by this pane: "Message".

Overridden methods:

- set

Custom methods:

- set_message

set_data(*data*)
Adds data to the pane's data dictionary.

Special key supported: "message" changes the displayed message.

set_message(*message*)
Change the message displayed in the pane.

class tkpane.lib.**NotebookPane**(*parent*, *pane_name*, *tab_specs*)
Create and populate a Tkinter Notebook widget.

Parameters

- **pane_name** – The name to be used to identify this pane in status messages.

- **tab_specs** – A list or tuple of two-element tuples; each two-element tuple contains the tab's label and a *build* function that is passed the Notebook widget and should populate the tab page with widgets and return the frame enclosing all widgets on that page.

This pane does not manage any data.

Name used by this pane: user-defined on initialization.

Overridden methods:

- set_style

Custom methods:

- notebook_widget

notebook_widget()
Return the Notebook widget.

tab_frame(*tab_name*)
Return the frame corresponding to the tab's name or label.

tab_frames()
Return a dictionary of tab names and the frame enclosing the contents for the tab.

tab_id(*tab_name*)
Return the tab ID (integer) corresponding to the tab's name or label.

class tkpane.lib.**OkCancelPane**(*parent*, *ok_action=None*, *cancel_action=None*)
Display OK and Cancel buttons.

There are no data keys specific to this pane.

Overridden methods:

- disable_pane
- enable_pane
- set_style
- focus

Custom methods:

- set_cancel_action
- set_ok_action
- ok
- cancel

cancel()
Trigger this pane's "Cancel" action.

focus()
Set the focus to the OK button.

ok()
Trigger this pane's "OK" action. The callback function will be passed this pane's data dictionary.

set_cancel_action(*cancel_action*)
Specify the callback function to be called when the "Cancel" button is clicked.

The callback function will not be passed any arguments.

set_ok_action(*ok_action*)
Specify the callback function to be called when the "OK" button is clicked.

The callback function should take a dictionary as an argument. It will be passed the OkCancelPane's entire data dictionary.

class tkpane.lib.**OutputDirPane**(*parent*, *optiondict=None*)
Get and display an output directory.

Parameters **optiondict** – a dictionary of option names and values for the Tkinter 'askdirectory' method (optional).

Data key managed by this pane: "output_dir".

Name used by this pane: "Output directory".

Overridden methods:

- entry_widgets
- valid_data
- save_data
- clear_pane
- disable_pane
- enable_pane
- set_style
- focus
- set_data

entry_widgets()
Return a list of widgets used for data entry.

focus()
Set the focus to the entry.

save_data(*is_valid*, *entry_widget*)
Update the pane's data dictionary with data from the entry widget.

Overrides TkPane class method.

set_data(*data*)
Update the pane's data dictionary.

Special key supported: 'directory' changes the directory name in the entry display.

set_key(*key_name*)
Change the name of the data key used for the entered data.

Parameters **key_name** – New name for the data key.

This method allows the name of the data key to be customized to eliminate conflicts with other OutputDirPane objects on the same UI.

valid_data(*widget=None*)
Return True or False indicating the validity of the directory entry.

Overrides TkPane class method.

class tkpane.lib.**OutputFilePane**(*parent*, *optiondict=None*)
Get and display an output filename.

Parameters **optiondict** – a dictionary of option names and values for the Tkinter 'asksaveasfilename' method (optional).

Data key managed by this pane: "output_filename".

Name used by this pane: "Output filename".

Overridden methods:

- entry_widgets
- valid_data
- save_data
- clear_pane
- disable_pane
- enable_pane
- set_style
- focus
- set_data

entry_widgets()
Return a list of widgets used for data entry.

focus()
Set the focus to the entry.

save_data(*is_valid*, *entry_widget*)
Update the pane's data dictionary with data from the entry widget.

Overrides TkPane class method.

set_data(*data*)
Update the pane's data dictionary with the provided data.

Special key supported: 'output_filename' changes the filename in the display.

set_key(*key_name*)
Change the name of the data key used for the entered data.

Parameters **key_name** – New name for the data key.

This method allows the name of the data key to be customized to eliminate conflicts with other OutputFilePane objects on the same UI.

valid_data(*widget=None*)
Return True or False indicating the validity of the filename entry.

Overrides TkPane class method.

class tkpane.lib.**PaneStyle**(*stylename*, *frame_config_dict={'padx': 6, 'pady': 6}*, *frame_grid_dict={}*)
Define a set of configuration options for frames and for widgets.

class tkpane.lib.**RadiobuttonPane**(*parent*, *pane_name*, *prompt*, *option_list*, *default_option=None*, *orient_vertical=True*, *button_action=None*, *key_name=None*, *config_opts=None*)
Display a Tkinter Radiobutton widget

Parameters

- **pane_name** – The name to be used to identify this pane in status messages.

- **prompt** – The text associated with the set of radiobuttons.
- **option_list** – List of radiobutton options, consisting of tuples in the format: (label, value).
- **default_option** – The label of the default option, e.g., "radioopt1" for ("radioopt", 1). If the default option is not actually present on the option list (or if no default option is specified), the default value is set to the empty string "".
- **orient_vertical** – Whether the radio button group should be oriented horizontally or vertically. Default is True (vertical)
- **button_action** – A callback to perform an action when a value is selected.
- **key_name** – The name to be used with the internal data dictionary to identify the entry data; use to avoid name conflicts with other RadiobuttonPane panes on the same UI (optional).
- **config_opts** – A dictionary of configuration options for the Radiobutton widget

Data keys managed by this pane: "radio" or the key name specified during initialization.

Name used by this pane: user-defined on initialization.

Overridden methods:

- entry widgets
- save_data
- clear_pane
- enable_pane
- disable_pane
- set_style
- set_data

Custom methods:

- set_key
- set_button_action
- do_button_action

do_button_action()
Trigger this pane's action. The callback function will be passed this pane's data dictionary.

save_data(*is_valid*, *entry_widget=None*)
Update the pane's data dictionary with data from the Radiobutton widget.

set_button_action(*button_action*)
Specify the callback function to be called when the button is clicked.

set_data(*data*)
Update the pane's data dictionary with the provided data.

Special key supported: 'prompt' changes the pane's prompt.

set_key(*key_name*)
Change the name of the data key used for the entered data.

Parameters **key_name** – New name for the data key.

This method allows the name of the data key to be customized to eliminate conflicts with other RadiobuttonPane objects on the same UI.

class tkpane.lib.**ScalePane**(*parent*, *pane_name*, *orientation*, *length*, *min_value*, *max_value*, *init_value*, *key_name=None*, *config_opts=None*)

Display a Tkinter Scale widget.

Parameters

- **pane_name** – The name to be used to identify this pane in status messages.
- **orientation** – "horizontal" for a horizontal scale bar, otherwise the scale bar will be vertical.
- **length** – Length of the scale bar, in pixels.
- **min_value** – Minimum value for the scale bar.
- **max_value** – Maximum value for the scale bar.
- **init_value** – Initial value for the scale bar.
- **key_name** – The name to be used with the internal data dictionary to identify the entry data; use to avoid name conflicts with other EntryPane objects on the same UI (optional).
- **config_opts** – A dictionary of scale widget configuration options

Data keys managed by this pane: "scale" or the key name specified during initialization.

Name used by this pane: user-defined on initialization.

This pane (the Scale widget) is considered to always have valid data.

Overridden methods:

- entry_widgets
- save_data
- enable_pane
- disable_pane
- set_style
- focus

Custom methods:

- scalewidget
- set_key

focus()
Set the focus to the entry.

save_data(*is_valid*, *entry_widget*)
Update the pane's data dictionary with data from the Scale widget.

scalewidget()
Returns the Scale widget.

set_key(*key_name*)
Change the name of the data key used for the entered data.

Parameters **key_name** – New name for the data key.

This method allows the name of the data key to be customized to eliminate conflicts with other ScalePane objects on the same UI.

class tkpane.lib.**ScaleSpinPane**(*parent*, *pane_name*, *prompt*, *min_value*, *max_value*, *init_value*, *length=None*, *key_name=None*, *scale_config_opts=None*, *spin_config_opts=None*)

Display a Scale and Spinbox, both displaying the same value.

Parameters

- **pane_name** – The name to be used to identify this pane in status messages.
- **prompt** – The prompt to be presented in a Label widget adjacent to the Scale widget.
- **min_value** – The minimum value allowed by the controls.
- **max_value** – The maximum value allowed by the controls
- **init_value** – The initial value displayed by the controls.
- **length** – The length of the Scale widget, in pixels (optional; default=100).
- **key_name** – The name to be used with the internal data dictionary to identify the entry data; use to avoid name conflicts with other ScaleSpinPane panes on the same UI (optional).

The Scale widget is horizontal, with the prompt to the left of it and the Spinbox to the right of it.

Overridden methods:

- entry_widgets
- enable_pane
- disable_pane
- set_style
- focus

Custom methods:

- scalewidget
- spinwidget
- set_key

disable_pane()
Disable the scale and spinbox widgets.

enable_pane()
Enable the scale and spinbox widgets.

focus()
Set the focus to the scale widget.

scalewidget()
Return the Scale widget.

set_key(*key_name*)
Change the name of the data key used for the entered data.

Parameters **key_name** – New name for the data key.

This method allows the name of the data key to be customized to eliminate conflicts with other ScalePane objects on the same UI.

set_style(*ttk_style*)
Change the style of the scale widget.

spinwidget()
Return the Spinbox widget.

class tkpane.lib.**SpinboxPane**(*parent*, *pane_name*, *prompt*, *min_value*, *max_value*, *key_name=None*, *optiondict=None*)
Display a Tkinter Spinbox widget with a prompt.

Parameters

- **pane_name** – The name to be used to identify this pane in status messages.
- **prompt** – The prompt to be presented in a Label widget adjacent to the entry.
- **min_value** – The minimum value that can be selected.
- **max_value** – The maximum value that can be selected.
- **key_name** – The name to be used with the internal data dictionary to identify the entry data; use to avoid name conflicts with other EntryPane objects on the same UI (optional).

Data keys managed by this pane: "spinbox" or the key name specified during initialization.

Name used by this pane: user-defined on initialization.

Overridden methods:

- entry_widgets
- valid_data
- save_data
- clear_pane
- enable_pane
- disable_pane
- focus
- set_data

Custom method:

- set_key

focus()
Set the focus to the entry.

save_data(*is_valid*, *entry_widget*)
Update the pane's data dictionary with data from the Spinbox widget.

set_data(*data*)
Update the pane's data dictionary with the provided data.

Special key supported: 'prompt' changes the pane's prompt.

set_key(*key_name*)
Change the name of the data key used for the entered data.

Parameters **key_name** – New name for the data key.

This method allows the name of the data key to be customized to eliminate conflicts with other SpinboxPane objects on the same UI.

class tkpane.lib.**StatusProgressPane**(*parent*)
Display a status bar and progress bar.

There are no data keys managed by this pane.

Overridden methods:

- clear_pane
- set_style
- values

Custom methods:

- set_status(message): Sets the status bar message.
- set_determinate(): Sets the progress bar to determinate mode.
- set_indeterminate(): Sets the progress bar to indeterminate mode.
- set_value(value): Sets a determinate progress bar to the specified value (0-100).
- start(): Starts an indefinite progress bar.
- stop(): Stops an indefinite progress bar.

clear_pane()
Clears the status bar and progress bar.

clear_status()
Clears the status bar message.

set_data(*data*)
Update the pane's data dictionary with the provided data.

Special keys supported: 'message' and 'value'. * message: New status message. * value: New value for the progress bar.

set_determinate()
Sets the progress bar to definite mode.

set_indeterminate()
Sets the progress bar to indefinite mode.

set_status(*message*)
Sets the status bar message.

set_value(*value*)
Sets the progress bar indicator.

The 'value' argument should be between 0 and 100, and will be trimmed to this range if it is not.

start()
Start an indefinite progress bar running.

stop()
Stop an indefinite progress bar.

class tkpane.lib.**TableDisplayPane**(*parent*, *message=None*, *column_headers=[]*, *rowset=[]*)
Display a specified data table.

Parameters

- **message** – A message to display above the data table.
- **column_headers** – A list of the column names for the data table.
- **rowset** – An iterable that yields lists of values to be used as rows for the data table.

There are no data keys managed by this pane.

Overridden methods:

- clear_pane
- set_data

Custom methods:

- display_data

display_data(*column_headers*, *rowset*)
Display a new data set on the pane.

Parameters

- **column_headers** – A list of strings for the headers of the data columns.
- **rowset** – A list of lists of data values to display. The outer list is rows, the inner lists are columns.

set_data(*data*)
Update the pane's data dictionary with the provided data.

Special keys supported: 'message' and 'table'.

- message: Text to replace the message above the table.
- table: the value should be a two-element tuple, of which the first element is a list of the column header names and the second is a list (rows) of lists (columns) containing the table's data.

class tkpane.lib.**TableSelectPane**(*parent*, *message=None*, *column_headers=[]*, *rowset=[]*)
Display a specified data table and allow a row to be selected.

Parameters

- **message** – A message to display above the data table.
- **column_headers** – A list of the column names for the data table.
- **rowset** – An iterable that yields lists of values to be used as rows for the data table.

Data key managed by this pane: "table_data".

- table_data: A dictionary of the selected data; keys are the table column names.

Name used by this pane: "Table select".

Overridden methods:

- valid_data
- save_data
- clear_pane
- focus
- set_data

Custom methods:

- display_data

display_data(*column_headers*, *rowset*)
Display a new data set on the pane.

Parameters

- **column_headers** – A list of strings for the headers of the data columns.

- **rowset** – A list of lists of data values to display. The outer list is rows, the inner lists are columns.

set_data(*data*)
Update the pane's data dictionary with the provided data.

Special keys supported: 'message' and 'table'.

- message: Text to replace the message above the table.
- table: the value should be a two-element tuple, of which the first element is a list of the column header names and the second is a list (rows) of lists (columns) containing the table's data.

set_key(*key_name*)
Change the name of the data key used for the entered data.

Parameters **key_name** – New name for the data key.

This method allows the name of the data key to be customized to eliminate conflicts with other EntryPane objects on the same UI.

class tkpane.lib.**TextPane**(*parent*, *key_name=None*, *optiondict=None*, *initial_text=None*, *required=False*, *blank_is_valid=False*)
Display a Tkinter Text widget.

Parameters

- **key_name** – The name to be used with the internal data dictionary to identify the text data; use to avoid name conflicts with other TextPane objecs on the same UI (optional).
- **optiondict** – A dictionary of option names and values for initial configuration of the Text widget (optional).
- **initial_text** – Initial contents for the Text widget (optional).
- **required** – A Boolean indicating whether valid data must be entered (optional; default is False).
- **blank_is_valid** – A Boolean indicating whether, if an entry is not required, a blank value should be treated as a valid entry (optional; default is False). If this is set to True, an empty string may be passed to any other pane that requires this pane.

Because of the large number of uses of the Text widget, this pane provides direct access to the Text widget via the 'textwidget' method. To simplify use, this pane also provides direct methods for appending to, replacing, and clearing the contents of the Text widget. The custom methods 'set_status' and 'clear_status' allow a TextPane to be used as a status_reporter callback for any other type of pane.

Data keys managed by this pane: "text" or the key name specified during initialization.

Name used by this pane: "Text".

Overridden methods:

- entry_widgets
- save_data
- valid_data
- clear_pane
- enable_pane
- disable_pane
- set_style

- focus
- set_data

Custom methods:

- textwidget
- replace_all
- append
- set_status
- clear_status
- set_key
- set_entry_validator

append(*more_text*, *scroll=True*)
Inserts the given text at the end of the Text widget's contents.

clear_status()
Clear the entire widget.

focus()
Set the focus to the text widget.

save_data(*is_valid*, *entry_widget*)
Update the pane's data dictionary with data from the text widget.

set_data(*data*)
Update the pane's data dictionary with the provided data.

Special keys supported: 'text' changes the contents of the text widget.

set_entry_validator(*fn*)
Set the callback function that will be used to check the entered value.

This function must take the entry value as an argument and return a Boolean.

set_key(*key_name*)
Change the name of the data key used for the text data.

Parameters **key_name** – New name for the text data key.

This method allows the name of the data key to be customized to eliminate conflicts with other TextPane objects on the same UI.

set_status(*status_msg*)
Inserts the status message at the end of the Text widget's contents.

set_style(*ttk_style*)
Sets the style of the scrollbars.

Note that the Text widget is not a ttk themed widget, and so no ttk style can be applied.

textwidget()
Return the text widget object, to allow direct manipulation.

class tkpane.lib.**UserPane**(*parent*)
Display a user's name and a button to prompt for a user's name and password.

Data keys managed by this pane: "name" and "password".

Name used by this pane: "User authorization".

Overridden methods:

- valid_data
- clear_pane
- send_status_message
- focus

Custom methods:

- set_user
- set_user_validator

focus()
Set the focus to the button.

send_status_message(*is_valid*)
Send a status message reporting data values and/or validity if data have changed.

set_keys(*user_key_name*, *password_key_name*)
Change the names of the data keys used for the entered data.

Parameters

- **user_key_name** – New name for the key for the user's name.
- **password_key_name** – New name for the key for the user's password.

This method allows the name of the data key to be customized to eliminate conflicts with other UserPane objects on the same UI.

set_user_validator(*fn*)
Set the callback function that will be used to check the entered user name and password.

This function must take the user name and password as arguments and return a Boolean.

valid_data(*widget=None*)
Return True or False indicating whether or not a name and password have been entered.

class tkpane.lib.**UserPasswordPane**(*parent*)
Display a user's name and a button to prompt for a user's name and password.

Data keys managed by this pane: "name" and "password".

Name used by this pane: "User credentials".

Overridden methods:

- valid_data
- save_data
- clear_pane
- enable_pane
- disable_pane
- send_status_message
- set_style
- focus

Custom method:

- set_user_validator

focus()
Set the focus to the user name.

save_data(*is_valid*, *entry_widget*)
Update the pane's data dictionary with data from the Entry widgets.

send_status_message(*is_valid*)
Send a status message reporting data values and/or validity if data have changed.

set_keys(*user_key_name*, *password_key_name*)
Change the names of the data keys used for the entered data.

Parameters

- **user_key_name** – New name for the key for the user's name.
- **password_key_name** – New name for the key for the user's password.

This method allows the name of the data key to be customized to eliminate conflicts with other UserPasswordPane objects on the same UI.

set_user_validator(*fn*)
Set the callback function that will be used to check the entered user name and password.

This function must take the user name and password as arguments and return a Boolean.

valid_data(*widget=None*)
Return True or False indicating whether or not a name and password have been entered.

CHAPTER 10

Examples

Following are three examples. The first illustrates the creation of a simple application using panes that are defined in `tkpane.lib`, the second illustrates the construction of a simple custom pane, and the third is a small working application to display selected columns from a CSV file.

10.1 Example 1: Using Panes from `tkpane.lib`

The following is a simple example that uses several of the panes in the `tkpane.lib` library. Dependencies are created so that both an input file name and an output file name must be specified before the "OK" button can be used.

This example uses the `tklayout` library to simplify the arrangement of the panes into an overall application UI.

```
try:
    import Tkinter as tk
except:
    import tkinter as tk
import tkpane.lib
import tklayout

# Add a method to the AppLayout class to get a pane: the first child
# of a frame's widgets.
def layout_pane(self, pane_name):
    return self.frame_widgets(pane_name)[0]
tklayout.AppLayout.pane = layout_pane


# Lay out the panes.  This example isn't focused on the tklayout
# library, so the layout is simple: just a vertically stacked set of
# panes.  The UI elements in the layout are all named, and each of
# these named elements will correspond to a pane.
layout = tklayout.AppLayout()
app = layout.column_elements(["infile_pane", "outfile_pane",
                              "text_pane", "entry_pane",
```

(continues on next page)

(continued from previous page)

```
                                "button_pane"],
                                row_weights=[0,0,1,0,0])

root = tk.Tk()
root.title("Demo of the TkPane Package")

# Use an extra frame within the root element with padding to add
# extra space around the outermost app widgets.
appframe = tk.Frame(root, padx=11, pady=11)
appframe.pack(expand=True, fill=tk.BOTH)

# Create the frames that implement the layout.
layout.create_layout(appframe, app)

# Use the pane class constructors to populate each UI element in the
# layout.  Panes from the library are used; no custom panes are
# created for this example.
layout.build_elements({"infile_pane": tkpane.lib.InputFilePane,
                   "outfile_pane": tkpane.lib.OutputFilePane,
                   "text_pane": tkpane.lib.TextPane,
                   "entry_pane":
                     lambda p: tkpane.lib.EntryPane(p, "comments",
                                                    "Comments:"),
                   "button_pane": tkpane.lib.OkCancelPane
                   })

# Get references to the actual pane objects so that they can be
# customized.
infile_pane = layout.pane("infile_pane")
outfile_pane = layout.pane("outfile_pane")
text_pane = layout.pane("text_pane")
button_pane = layout.pane("button_pane")

# Require an input file name and output file name to be entered for
# the 'OK' button to be enabled.
button_pane.requires(infile_pane)
button_pane.requires(outfile_pane)
# For this example, only the button_pane needs something disabled
# (the OK button), but several panes are included below to
# illustrate functionality.
tkpane.en_or_dis_able_all([infile_pane, outfile_pane, button_pane])

# Make the infile and outfile panes report their status changes to
# the text pane.
infile_pane.status_reporter = text_pane
outfile_pane.status_reporter = text_pane

# Disable user entry into the text pane; it is used only as a status
# log.
text_pane.disable()

# Make the buttons report their activation.
def ok_click(*args):
    # When this is bound, it will receive event arguments, which
    # can be ignored.
    text_pane.set_status("OK button clicked.")
button_pane.set_ok_action(ok_click)
```

(continues on next page)

(continued from previous page)

```
79
80  def cancel_click(*args):
81      text_pane.set_status("Cancel button clicked.")
82  button_pane.set_cancel_action(cancel_click)
83
84  # Bind <Enter> and <Esc> to the buttons.
85  root.bind("<Return>", ok_click)
86  root.bind("<Escape>", cancel_click)
87
88
89  # Run the application
90  root.mainloop()
```

This will produce an application that looks like Figure 3, below, after an input file name has been entered.

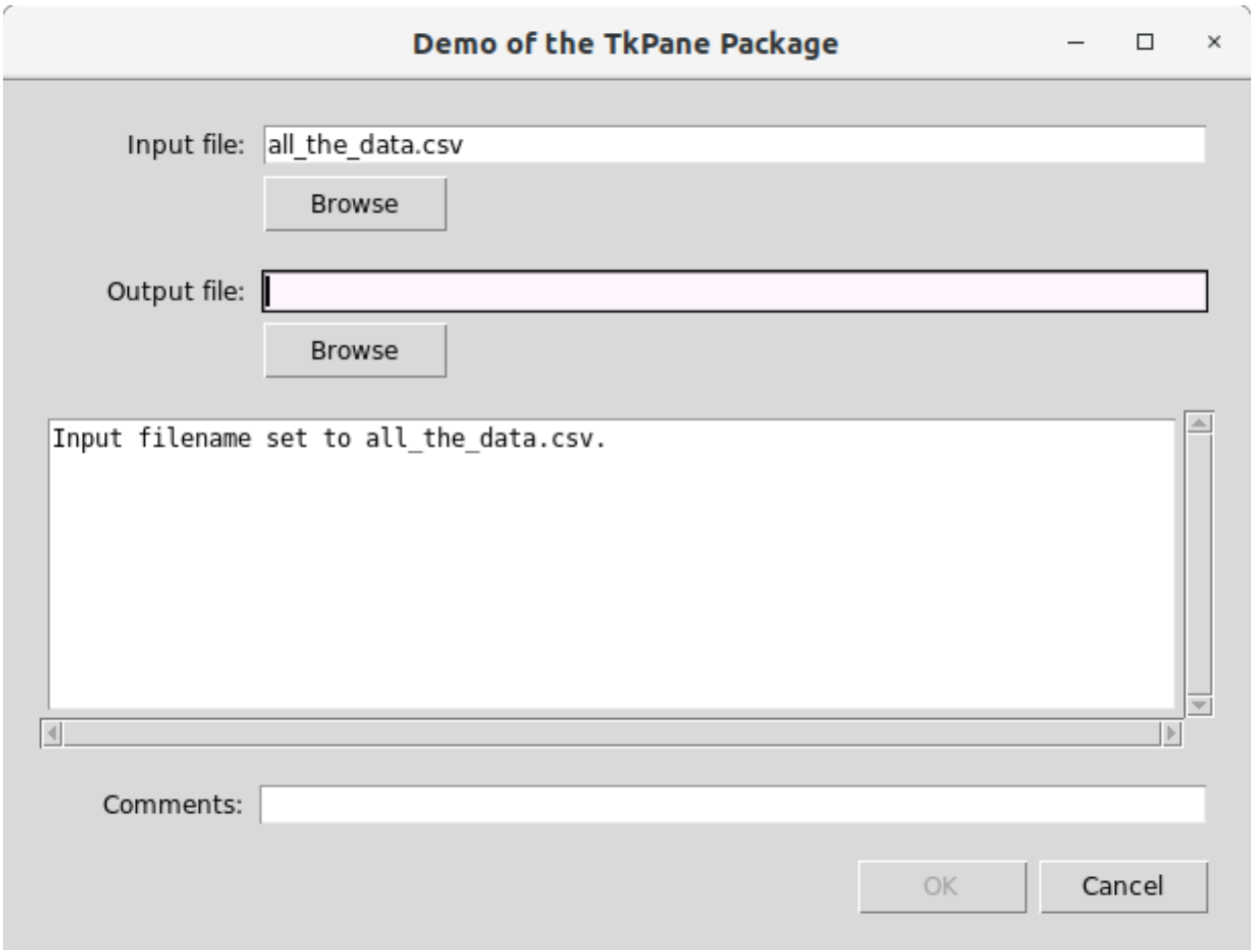

Fig. 1: Figure 3. Example Pane Usage

10.2 Example 2. Construction of a Custom Pane

Although the panes in `tkpane.lib` are useful, many application will need to subclass the `TkPane` class to create new custom panes. Creating a custom pane that handles user entries or other data may require any or all of the

following, in addition to the creation of the widgets on the pane:

- Overriding some methods of the `TkPane` class
- Setting some class attributes.
- Adding a trace on Tkinter variables or a <Key> binding on widgets to allow data validation.
- Adding custom methods specific to the purpose of the pane.

The first three of these are illustrated in the following code, which creates a simple pane containing a Tkinter Entry widget. This is a slightly modified version of the `EntryPane` class from `tkpane.lib`.

```
class EntryPane(tkpane.TkPane):
    def __init__(self, parent, pane_name, prompt, key_name=None):
        tkpane.TkPane.__init__(self, parent, pane_name, {}, {})
        self.datakeyname = "entry" if key_name is None else key_name
        self.datakeylist = [self.datakeyname]
        self.prompt = ttk.Label(self, text=prompt, width=max(12, len(prompt)),␣
↪anchor=tk.E)
        self.entry_var = tk.StringVar()
        if tkpane.use_ttk:
            self.entrywidget = ttk.Entry(self, textvariable=self.entry_var,␣
↪exportselection=False)
            self.widget_type = "ttk"
        else:
            self.entrywidget = tk.Entry(self, textvariable=self.entry_var,␣
↪exportselection=False)
            self.widget_type = "tk"
        self.prompt.grid(row=0, column=0, padx=3, pady=3, sticky=tk.EW)
        self.entrywidget.grid(row=0, column=1, padx=3, pady=3, sticky=tk.EW)
        self.rowconfigure(0, weight=1)
        self.columnconfigure(0, weight=0)
        self.columnconfigure(1, weight=1)
        parent.rowconfigure(0, weight=1)
        parent.columnconfigure(0, weight=1)
        self.entry_var.trace("w", self.check_entrychange)

    #---------------------------------------------------------------------------
    #   Overrides of class methods.
    #...........................................................................

    def entry_widgets(self):
        return [self.entrywidget]

    def valid_data(self, entry_widget=None):
        text = self.entry_var.get()
        return not (text == "" and self.required)

    def save_data(self, is_valid, entry_widget):
        """Update the pane's data dictionary with data from the Entry widget."""
        text = self.entry_var.get()
        if is_valid:
            self.datadict[self.datakeyname] = text
        else:
            self.clear_own()

    def clear_pane(self):
        self.entry_var.set("")

```

(continues on next page)

(continued from previous page)

```
    def enable_pane(self):
        self._enablewidgets([self.prompt, self.entrywidget])

    def disable_pane(self):
        self._disablewidgets([self.prompt, self.entrywidget])

    def set_style(self, ttk_style):
        self._setstyle([self.prompt, self.entrywidget], ttk_style)

    def focus(self):
        """Set the focus to the entry."""
        self.entrywidget.focus_set()

    def set_data(self, data):
        """Update the pane's data dictionary with the provided data.

        Special key supported: 'prompt' changes the pane's prompt.
        """
        spkey = "prompt"
        if spkey in data:
            self.prompt.configure(text=data[spkey])
        self.set_allbut(data, [spkey])

    #---------------------------------------------------------------------------
    #   Custom methods.
    #...........................................................................

    def check_entrychange(self, *args):
        self.handle_change_validity(self.valid_data(None), self.entrywidget)

    def set_key(self, key_name):
        """Change the name of the data key used for the entered data.

        :param key_name: New name for the data key.

        This method allows the name of the data key to be customized to
        eliminate conflicts with other EntryPane objects on the same UI.
        """
        if self.datakeyname in self.datadict:
            self.datadict[key_name] = self.datadict[self.datakeyname]
            del self.datadict[self.datakeyname]
        self.datakeyname = key_name
        self.datakeylist = [key_name]
```

The arguments for the `__init__` constructor method for this class are:

- *parent:* The Tkinter widget that will be the parent for this pane. All `TkPane` subclasses must take this argument and pass it on to the `TkPane` class's own constructor method.
- *pane_name:* A text string used to identify the pane in automatically-generated status messages. Some `TkPane` subclasses may assign this themselves, but in this case it is provided by the user in case multiple `EntryPane` objects are used, and they are to be distinguished in status messages.
- *prompt:* This is the text that will be displayed in a Tkinter Label widget adjacent to the Entry widget.
- *key_name:* A substitute for the default data key name of "entry".

The `__init__` method for a custom pane must call the constructor method for the `TkPane` class itself. The arguments for this call are:

- *parent:* The Tkinter widget that will be the parent for this pane.
- *pane_name:* A text string used to identify the pane in status messages.
- *config_opts:* A dictionary of Tkinter configuration option that will be applied to the Frame widget that encloses this pane's widgets. This argument is optional.
- *grid_opts:* A dictionary of Tkinter options for the `grid()` geometry manager that will be applied to this pane's frame when it is placed within its parent. By default, a pane's frame is made sticky to the N, S, E, and W, and made resizeable in both vertical and horizontal directions. This argument is optional.

The `__init__` method also assigns a value to `self.datakeylist`. This overrides a `TkPane` class attribute. The `datakeylist` should be a list of the dictionary keys for all data values managed by this pane. If this list is not specified, the automatic data-handling procedures of the `TkPane` class will not operate properly.

The `__init__` method for this pane class does not set the pane's `required` attribute. The constructors for other pane classes might (as illustrated in *Example 1*). The `required` attribute of objects of this class might be set after instantiation, either directly by the user or by a `requires()` method call from another pane that uses an object of this pane class as an argument.

Most of the remainder of the `__init__` method simply creates the widgets for this pane, but one additional element that is key to the proper functioning of the pane is the `trace()` method applied to the Tkinter variable that contains the entry value. This method, or an equivalent binding to a <Key> event for the widget itself, is essential for keystroke-by-keystroke validation of data changes, and propagation of *enable* and *disable* actions to related panes. The trace calls a custom method of this class, `check_entrychange()`, that accepts the arguments provided by the `trace()` method and simply chains to the `TkPane` class method `handle_change_validity()`. As the word "handle" in the name of this latter method implies, the method calls all the CbHandler callbacks to enable or disable other panes as necessary.

Because this custom class handles data entry, it overrides the six `TkPane` class methods that should, as a rule, be customized by any `TkPane` subclass that manages data:

- `entry_widgets()`: This method returns a list of all of the widgets on the pane that accept data entry and for which those data may be either valid or invalid.
- `valid_data()`: This method returns a Boolean indicating whether the data in the specified widget, or in all widgets if no widget is provided as an argument, are valid. In this case, if the `required` attribute of the pane is set, any non-empty entry is considered valid, and only an empty entry is considered invalid.
- `save_data()`: This method revises the pane's data dictionary to reflect the data on the pane's widgets. If no widget is provided as an argument, it should address all widgets containing data. If the data are valid, they are added to the pane's data dictionary; if the data are invalid, the data and key are removed from the pane's data dictionary. This method calls the `clear_data()` method of the `TkPane` class to remove all pane-specific data from the data dictionary.
- `clear_pane()`: This method removes any data from the widget. In this case the Entry widget is linked to a Tkinter StringVar, so the contents of the StringVar are set to an empty string.
- `enable_pane()`: This method ensures that the user is able to interact with the widget(s) on the pane.
- `disable_pane()`: This method ensures that the user is not able to interact with the widget(s) on the pane.

Although other custom pane classes may have different widgets and additional custom methods, this simple `TkPane` subclass is a template for all custom data-handling pane classes.

10.3 Example 3. A Simple CSV Explorer Application

This is a simple application that allows a CSV file to be selected, then displays a list of the column names in that file, and after one or more columns has been selected, displays the data for those columns.

Three panes from `tkpane.lib` are used:

- `InputFilePane` to get the CSV file name.
- `ListboxPane` to get the columns to display
- `TableDisplayPane` to display the selected columns of the CSV file.

Because populating the table display is potentially time-consuming, the table is not populated until the user leaves the list box used to select columns. This is accomplished using the `enable_on_other_exit_only` argument of the `requires()` method.

```
import csv
import sqlite3
try:
    import Tkinter as tk
except:
    import tkinter as tk
import tkpane
import tkpane.lib
import tklayout


class MemDb(object):
    # An in-memory SQLite database for a single data table named "src".
    def __init__(self):
        self.conn = sqlite3.connect(":memory:")
        self.fileread = None
        self.column_names = None
        self.tablename = "src"
    def quote_str(self, str):
        """Add single quotes around a string."""
        if len(str) == 0:
            return "''"
        if len(str) == 1:
            if str == "'":
                return "''''"
            else:
                return "'%s'" % str
        if str[0] != "'" or str[-1:] != "'":
            return "'%s'" % str.replace("'", "''")
        return str
    def quote_list(self, l):
        # Add single quotes around all strings in the list.
        return [self.quote_str(x) for x in l]
    def quote_list_as_str(self, l):
        # Convert a list of strings to a single string of comma-delimited, quoted␣
↪tokens.
        return ",".join(self.quote_list(l))
    def read_csv(self, data_fn):
        if self.fileread is None or data_fn != self.fileread:
            dialect = csv.Sniffer().sniff(open(data_fn, "rt").readline())
            inf = csv.reader(open(data_fn, "rt"), dialect)
            self.column_names = inf.next()
            colstr = ",".join(self.column_names)
            try:
                self.conn.execute("drop table %s;" % self.tablename)
            except:
                pass
```

(continues on next page)

(continued from previous page)

```
            self.conn.execute("create table %s (%s);" % (self.tablename, colstr))
            for l in inf:
                sql = "insert into %s values (%s);" % (self.tablename, self.quote_
↪list_as_str(l))
                self.conn.execute(sql)
                self.conn.commit()
    def rows(self, columnlist):
        # Return the rows, as a list of lists, for the specified columns.
        colspec = ",".join(columnlist)
        sql = "select %s from %s;" % (colspec, self.tablename)
        return self.conn.execute(sql).fetchall()


def set_listbox_contents(listbox_pane, listbox_ddict):
    # This function is meant to be called from the 'enable()' method
    # of the listbox (i.e., added to its 'on_enable' callback list),
    # The data dictionary should contain an "input_filename"
    # value, which will be read into internal storage, and the headers
    # used to populate the listbox.
    fn = listbox_ddict["input_filename"]
    db = listbox_ddict["db"]
    db.read_csv(fn)
    listbox_pane.set_newitems(db.column_names)

def populate_table(table_pane, table_ddict):
    # This function is meant to be called from the 'enable()' method
    # of the table display (i.e., added to its 'on_enable' callback list).
    headerlist = table_ddict["listbox"]
    db = table_ddict["db"]
    datarows = db.rows(headerlist)
    table_pane.display_data(headerlist, datarows)


def main():

    # Create a context for variables to be shared.
    sharedvars = {}

    #--------------  Model  -----------------
    # Create the database connection for the CSV file and
    # put it in the shared context.
    sharedvars["db"] = MemDb()

    #--------------  View  -----------------
    # Lay out the panes.
    layout = tklayout.AppLayout()
    # A row with 1) a listbox to select column headers,  and
    #            2) a table display for the table.
    displays = layout.row_elements(["headerlist", "table"], column_weights=[1,2])
    # A prompt for an input file above the listbox and table.
    app = layout.column_elements(["infile_pane", displays], row_weights=[0,1])

    root = tk.Tk()
    root.title("CSV File Explorer")

    # Use an extra frame within the root element with padding to add extra space
    # around the outermost app widgets.
```

(continues on next page)

(continued from previous page)

```
    appframe = tk.Frame(root, padx=9, pady=9)
    appframe.pack(expand=True, fill=tk.BOTH)

    panes = tkpane.build_ui(layout, appframe, app, {
             "infile_pane": lambda p: tkpane.lib.InputFilePane(p,
                            optiondict={"filetypes": (("CSV files", "*.csv"),)}),
             "headerlist":  lambda p: tkpane.lib.ListboxPane(p,
                            "headers", [], width=10),
             "table":       lambda p: tkpane.lib.TableDisplayPane(p,
                            message="Mouse over the table to refresh.")
             })

    # Set dependencies among panes.
    panes["headerlist"].requires(panes["infile_pane"])
    panes["table"].requires(panes["headerlist"], enable_on_other_exit_only=True,␣
↪clear_on_disable=True)

    #--------------  Controller  -----------------
    # Set custom callbacks to populate the listbox and table display.
    panes["headerlist"].on_enable.append(tkpane.PaneAllDataHandler(set_listbox_
↪contents))
    panes["table"].on_enable.append(tkpane.PaneAllDataHandler(populate_table))

    # Give the shared variable context to the listbox and table panes
    # so they (their controller functions) can access the data (model).
    panes["headerlist"].set_data(sharedvars)
    panes["table"].set_data(sharedvars)

    #--------------  Run  -----------------
    root.mainloop()


main()
```

As this example illustrates, if suitable pane classes are available, a Tkinter UI can be assembled, and its elements linked together, without any direct use of Tkinter objects or methods.

CHAPTER 11

Notes

1. Every pane that manages data must have a dictionary key (or keys) to identify the data value(s). Every pane class in `tkpane.lib` has a default name (or names) for these data key(s). When multiple panes of the same class are on the same UI, their data key names will coincide, and this may be a conflict if any other pane (or application code) uses the data dictionary from more than one of those panes. Each pane class therefore has a `set_key()` or `set_keys()` method that allows the data key(s) to be changed. Some pane classes also accept a new data key name as an initialization parameter. If the data keys of any pane must be changed when the UI is created, those changes must be made *before* any dependencies are established with the `requires()` or `can_use()` methods.
2. The `status_reporter` attribute of a pane, if used, must be set to an object (e.g., another pane) that has a `set_status()` method and, if needed, a `clear_status()` method. The `StatusProgressPane` and `TextPane` classes in `tkpane.lib` have these methods.
3. The `progress_reporter` attribute of a pane, if used, must be set to an object (e.g., another pane) that has the following methods:
    - `set_determinate()`: Sets the progress bar to alter the progress display only in response to calls to the `set_value()` method.
    - `set_indeterminate()`: Sets the progress bar to a continuously active display, indicating that action is underway.
    - `set_value()`: Changes the display of a determinate progress bar.
    - `start()`: Starts an indeterminate status bar.
    - `stop()`: Stops and indeterminate status bar.

    The `StatusProgressPane` class in `tkpane.lib` has these methods.

CHAPTER 12

Availability

The TkPane library is available on PyPi. It can be installed with:

```
pip install tkpane
```

The latest code is available from the Bibucket repository.

CHAPTER 13

Related Software

TkLayout (code and documentation) The TkLayout Python package simplifies the construction of a Tkinter interface by allowing the developer to describe the structure of the UI from the inside out, as successive nestings of rows and columns of elements, and then to create all of the Tkinter frames to implement this structure with one command. Although `tklayout` and `tkpane` can be used entirely independently, they work together well because panes can easily be used to fill the UI elements defined in the layout. Examples *1* and *3* both show the use of `tklayout` with `tkpane`.

CHAPTER 14

Copyright and License

CHAPTER 15

Contributors

Elizabeth Shea Assistance with the conceptualization of pane interactions, radio button pane, original versions of `UserPane` and `OutputDirPane`, and rigorous testing.

CHAPTER 16

Change Log

Date	Version	Revision
2018-03-17	1.0.0	Revised documentation.
2018-03-14	0.30.0	Modified EntryPane; added parameters 'required' and 'blank_is_valid'. Modified RadiobuttonPane to allow either orientation. Initialized datadict in panes with default values. Added function 'run_validity_callbacks()'.
2018-03-12	0.27.0	Modified title of UserPane in lib.py.
2018-03-10	0.26.0	Added 'set_entry_validator()' method to EntryPane in lib.py. Changed invalid colors to be configurable at module level. Changed use of ttk or tk widgets to be configurable at the module level.
2018-03-01	0.22.0	Added 'requires_datavalue() and 'clear_on_disable()' methods. Improved support for 'on_save_change' callback list. Added 'set_filename_validator()' method to InputFilePane.
2018-02-27	0.19.0	Added 'on_save_change' callback list. Added missing 'entry_widgets()' method to UserPasswordPane.
2018-02-27	0.18.0	Backwards-incompatible change since 0.14.0: changed object returned by 'build_ui()' to a dict. Added 'tab_frames()' method to NotebookPane. Set the CheckboxPane's value to False even when False is considered an invalid state.
2018-02-22	0.16.1	Changed method to set 'invalid' color for ttk widgets; correction to 'can_use()' method.
2018-02-21	0.15.0	Added parameter 'disable_on_other_exit_only' to the 'requires()' method.
2018-02-21	0.14.0	Modified internal callback methods, corrected Python 3 compatibility in status reporting, added 'set_key()' methods in tkpane.lib where missing, and added 'build_ui()'.
2018-02-19	0.11.0	Added TableSelectPane to lib.
2018-02-19	0.10.0	Backwards-incompatible changes to 'clear_data()'; added 'clear_own()'. Added generic panes to lib.

CHAPTER 17

Index

- search

Python Module Index

t

tkpane, 21
tkpane.lib, 27

Index

A

all_data() (tkpane.TkPane method), 22
AllDataHandler (class in tkpane), 26
append() (tkpane.lib.TextPane method), 46

B

build_ui() (in module tkpane), 27
ButtonPane (class in tkpane.lib), 27

C

can_enable() (tkpane.TkPane method), 22
can_use() (tkpane.TkPane method), 22
cancel() (tkpane.lib.OkCancelPane method), 36
canvas_widget() (tkpane.lib.CanvasPane method), 28
CanvasPane (class in tkpane.lib), 28
CbHandler (class in tkpane), 26
CheckboxPane (class in tkpane.lib), 29
clear() (tkpane.TkPane method), 22
clear_data() (tkpane.TkPane method), 23
clear_on_disable() (tkpane.TkPane method), 23
clear_own() (tkpane.TkPane method), 23
clear_pane() (tkpane.lib.CanvasPane method), 28
clear_pane() (tkpane.lib.StatusProgressPane method), 43
clear_pane() (tkpane.TkPane method), 23
clear_status() (tkpane.lib.StatusProgressPane method), 43
clear_status() (tkpane.lib.TextPane method), 46
ComboboxPane (class in tkpane.lib), 30

D

disable() (tkpane.TkPane method), 23
disable_pane() (tkpane.lib.CanvasPane method), 28
disable_pane() (tkpane.lib.ScaleSpinPane method), 41
disable_pane() (tkpane.TkPane method), 23
display_data() (tkpane.lib.TableDisplayPane method), 44
display_data() (tkpane.lib.TableSelectPane method), 44
do_button_action() (tkpane.lib.ButtonPane method), 28
do_button_action() (tkpane.lib.RadiobuttonPane method), 39

E

EmptyPane (class in tkpane.lib), 31
enable() (tkpane.TkPane method), 23
enable_or_disable_all() (in module tkpane), 26
enable_pane() (tkpane.lib.CanvasPane method), 28
enable_pane() (tkpane.lib.ScaleSpinPane method), 41
enable_pane() (tkpane.TkPane method), 23
entering() (tkpane.TkPane method), 23
entry_widgets() (tkpane.lib.InputFilePane method), 32
entry_widgets() (tkpane.lib.InputFilePane2 method), 33
entry_widgets() (tkpane.lib.OutputDirPane method), 37
entry_widgets() (tkpane.lib.OutputFilePane method), 38
entry_widgets() (tkpane.TkPane method), 23
EntryPane (class in tkpane.lib), 31

F

focus() (tkpane.lib.ButtonPane method), 28
focus() (tkpane.lib.CanvasPane method), 28
focus() (tkpane.lib.CheckboxPane method), 29
focus() (tkpane.lib.ComboboxPane method), 30
focus() (tkpane.lib.EntryPane method), 31
focus() (tkpane.lib.InputFilePane method), 32
focus() (tkpane.lib.InputFilePane2 method), 33
focus() (tkpane.lib.ListboxPane method), 35
focus() (tkpane.lib.OkCancelPane method), 36
focus() (tkpane.lib.OutputDirPane method), 37
focus() (tkpane.lib.OutputFilePane method), 38
focus() (tkpane.lib.ScalePane method), 40
focus() (tkpane.lib.ScaleSpinPane method), 41
focus() (tkpane.lib.SpinboxPane method), 42
focus() (tkpane.lib.TextPane method), 46
focus() (tkpane.lib.UserPane method), 47
focus() (tkpane.lib.UserPasswordPane method), 48
focus() (tkpane.TkPane method), 24

H

handle_change_validity() (tkpane.TkPane method), 24
handle_exit_validity() (tkpane.TkPane method), 24

I

InputFilePane (class in tkpane.lib), 32
InputFilePane2 (class in tkpane.lib), 33

L

layout_panes() (in module tkpane), 26
leaving() (tkpane.TkPane method), 24
ListboxPane (class in tkpane.lib), 34

M

MessagePane (class in tkpane.lib), 35

N

notebook_widget() (tkpane.lib.NotebookPane method), 36
NotebookPane (class in tkpane.lib), 35

O

ok() (tkpane.lib.OkCancelPane method), 36
OkCancelPane (class in tkpane.lib), 36
OutputDirPane (class in tkpane.lib), 37
OutputFilePane (class in tkpane.lib), 37

P

PaneDataHandler (class in tkpane), 26
PaneKeyHandler (class in tkpane), 26
PaneStyle (class in tkpane.lib), 38

R

RadiobuttonPane (class in tkpane.lib), 38
report_progress() (tkpane.TkPane method), 24
report_status() (tkpane.TkPane method), 24
requires() (tkpane.TkPane method), 24
requires_datavalue() (tkpane.TkPane method), 25
run_validity_callbacks() (in module tkpane), 26

S

save_data() (tkpane.lib.CheckboxPane method), 29
save_data() (tkpane.lib.ComboboxPane method), 30
save_data() (tkpane.lib.EntryPane method), 31
save_data() (tkpane.lib.InputFilePane method), 32
save_data() (tkpane.lib.InputFilePane2 method), 34
save_data() (tkpane.lib.ListboxPane method), 35
save_data() (tkpane.lib.OutputDirPane method), 37
save_data() (tkpane.lib.OutputFilePane method), 38
save_data() (tkpane.lib.RadiobuttonPane method), 39
save_data() (tkpane.lib.ScalePane method), 40
save_data() (tkpane.lib.SpinboxPane method), 42
save_data() (tkpane.lib.TextPane method), 46
save_data() (tkpane.lib.UserPasswordPane method), 48
save_data() (tkpane.TkPane method), 25
ScalePane (class in tkpane.lib), 39
ScaleSpinPane (class in tkpane.lib), 40
scalewidget() (tkpane.lib.ScalePane method), 40
scalewidget() (tkpane.lib.ScaleSpinPane method), 41
send_status_message() (tkpane.lib.UserPane method), 47
send_status_message() (tkpane.lib.UserPasswordPane method), 48
send_status_message() (tkpane.TkPane method), 25
set_button_action() (tkpane.lib.ButtonPane method), 28
set_button_action() (tkpane.lib.RadiobuttonPane method), 39
set_cancel_action() (tkpane.lib.OkCancelPane method), 36
set_data() (tkpane.lib.ButtonPane method), 28
set_data() (tkpane.lib.CheckboxPane method), 29
set_data() (tkpane.lib.ComboboxPane method), 30
set_data() (tkpane.lib.EntryPane method), 31
set_data() (tkpane.lib.InputFilePane method), 32
set_data() (tkpane.lib.InputFilePane2 method), 34
set_data() (tkpane.lib.MessagePane method), 35
set_data() (tkpane.lib.OutputDirPane method), 37
set_data() (tkpane.lib.OutputFilePane method), 38
set_data() (tkpane.lib.RadiobuttonPane method), 39
set_data() (tkpane.lib.SpinboxPane method), 42
set_data() (tkpane.lib.StatusProgressPane method), 43
set_data() (tkpane.lib.TableDisplayPane method), 44
set_data() (tkpane.lib.TableSelectPane method), 45
set_data() (tkpane.lib.TextPane method), 46
set_data() (tkpane.TkPane method), 25
set_determinate() (tkpane.lib.StatusProgressPane method), 43
set_entry_validator() (tkpane.lib.EntryPane method), 32
set_entry_validator() (tkpane.lib.TextPane method), 46
set_filename_validator() (tkpane.lib.InputFilePane method), 33
set_filename_validator() (tkpane.lib.InputFilePane2 method), 34
set_indeterminate() (tkpane.lib.StatusProgressPane method), 43
set_invalid_color() (tkpane.TkPane method), 25
set_key() (tkpane.lib.CheckboxPane method), 29
set_key() (tkpane.lib.ComboboxPane method), 30
set_key() (tkpane.lib.EntryPane method), 32
set_key() (tkpane.lib.InputFilePane method), 33
set_key() (tkpane.lib.InputFilePane2 method), 34
set_key() (tkpane.lib.ListboxPane method), 35
set_key() (tkpane.lib.OutputDirPane method), 37
set_key() (tkpane.lib.OutputFilePane method), 38
set_key() (tkpane.lib.RadiobuttonPane method), 39
set_key() (tkpane.lib.ScalePane method), 40
set_key() (tkpane.lib.ScaleSpinPane method), 41
set_key() (tkpane.lib.SpinboxPane method), 42
set_key() (tkpane.lib.TableSelectPane method), 45
set_key() (tkpane.lib.TextPane method), 46
set_keys() (tkpane.lib.UserPane method), 47
set_keys() (tkpane.lib.UserPasswordPane method), 48

set_message() (tkpane.lib.MessagePane method), 35
set_newitems() (tkpane.lib.ComboboxPane method), 31
set_newitems() (tkpane.lib.ListboxPane method), 35
set_ok_action() (tkpane.lib.OkCancelPane method), 36
set_scrolling() (tkpane.lib.CanvasPane method), 28
set_status() (tkpane.lib.StatusProgressPane method), 43
set_status() (tkpane.lib.TextPane method), 46
set_style() (tkpane.lib.ListboxPane method), 35
set_style() (tkpane.lib.ScaleSpinPane method), 41
set_style() (tkpane.lib.TextPane method), 46
set_style() (tkpane.TkPane method), 25
set_user_validator() (tkpane.lib.UserPane method), 47
set_user_validator() (tkpane.lib.UserPasswordPane method), 48
set_value() (tkpane.lib.StatusProgressPane method), 43
show_widget_validity() (tkpane.TkPane method), 25
show_widgets_validity() (tkpane.TkPane method), 26
SpinboxPane (class in tkpane.lib), 42
spinwidget() (tkpane.lib.ScaleSpinPane method), 41
start() (tkpane.lib.StatusProgressPane method), 43
StatusProgressPane (class in tkpane.lib), 42
stop() (tkpane.lib.StatusProgressPane method), 43

T

tab_frame() (tkpane.lib.NotebookPane method), 36
tab_frames() (tkpane.lib.NotebookPane method), 36
tab_id() (tkpane.lib.NotebookPane method), 36
TableDisplayPane (class in tkpane.lib), 43
TableSelectPane (class in tkpane.lib), 44
TextPane (class in tkpane.lib), 45
textwidget() (tkpane.lib.TextPane method), 46
TkPane (class in tkpane), 21
tkpane (module), 21
tkpane.lib (module), 27

U

UserPane (class in tkpane.lib), 46
UserPasswordPane (class in tkpane.lib), 47

V

valid_data() (tkpane.lib.CheckboxPane method), 30
valid_data() (tkpane.lib.InputFilePane method), 33
valid_data() (tkpane.lib.InputFilePane2 method), 34
valid_data() (tkpane.lib.ListboxPane method), 35
valid_data() (tkpane.lib.OutputDirPane method), 37
valid_data() (tkpane.lib.OutputFilePane method), 38
valid_data() (tkpane.lib.UserPane method), 47
valid_data() (tkpane.lib.UserPasswordPane method), 48
valid_data() (tkpane.TkPane method), 26
values() (tkpane.TkPane method), 26