Messagebox リファレンス

メッセージダイアログを表示する `tkinter.messagebox` モジュールについての詳細なリファレンスです。

概要

`tkinter.messagebox` モジュールは、ユーザーに情報を表示したり、確認を求めたり、警告を通知するためのダイアログボックスを提供します。インフォメーション、警告、エラー、質問など、様々な種類のメッセージダイアログを簡単に表示できます。

基本的な使用方法

シンプルなメッセージダイアログ

```
 import tkinter as tk
from tkinter import messagebox

def show_info():
    messagebox.showinfo("情報", "これは情報メッセージです。")

def show_warning():
    messagebox.showwarning("警告", "これは警告メッセージです。")

def show_error():
    messagebox.showerror("エラー", "これはエラーメッセージです。")

app = tk.Tk()
app.title("Messageboxの例")
app.geometry("300x200")

tk.Button(app, text="情報を表示", command=show_info).pack(pady=10)
tk.Button(app, text="警告を表示", command=show_warning).pack(pady=10)
tk.Button(app, text="エラーを表示", command=show_error).pack(pady=10)

app.mainloop()
```

クラスベースでのメッセージダイアログ

```
 import tkinter as tk
from tkinter import messagebox

class MessageboxApp(tk.Tk):
    def __init__(self):
        super().__init__()

        self.title("Messageboxの例（クラスベース）")
        self.geometry("300x350")

        self.create_widgets()

    def create_widgets(self):
        tk.Button(self, text="情報を表示", command=self.show_info).pack(pady=10)
        tk.Button(self, text="警告を表示", command=self.show_warning).pack(pady=10)
        tk.Button(self, text="エラーを表示", command=self.show_error).pack(pady=10)
        tk.Button(self, text="確認ダイアログ", command=self.show_question).pack(pady=10)

    def show_info(self):
        messagebox.showinfo("情報", "これは情報メッセージです。")

    def show_warning(self):
        messagebox.showwarning("警告", "これは警告メッセージです。")

    def show_error(self):
        messagebox.showerror("エラー", "これはエラーメッセージです。")

    def show_question(self):
        result = messagebox.askyesno("確認", "本当に実行しますか？")
        if result:
            messagebox.showinfo("結果", "「はい」が選択されました。")
        else:
            messagebox.showinfo("結果", "「いいえ」が選択されました。")

if __name__ == "__main__":
    app = MessageboxApp()
    app.mainloop()
```

主要な関数

情報表示系

関数	説明	戻り値
`showinfo(title, message, **options)`	情報アイコン付きのメッセージを表示します。	`'ok'`
`showwarning(title, message, **options)`	警告アイコン付きのメッセージを表示します。	`'ok'`
`showerror(title, message, **options)`	エラーアイコン付きのメッセージを表示します。	`'ok'`

質問・確認系

関数	説明	戻り値
`askquestion(title, message, **options)`	「はい」「いいえ」ボタン付きの質問ダイアログを表示します。	`'yes'` または `'no'`
`askyesno(title, message, **options)`	「はい」「いいえ」ボタン付きの確認ダイアログを表示します。	`True` または `False`
`askokcancel(title, message, **options)`	「OK」「キャンセル」ボタン付きの確認ダイアログを表示します。	`True` または `False`
`askretrycancel(title, message, **options)`	「再試行」「キャンセル」ボタン付きのダイアログを表示します。	`True` または `False`
`askyesnocancel(title, message, **options)`	「はい」「いいえ」「キャンセル」ボタン付きのダイアログを表示します。	`True`, `False`, または `None`

オプションパラメータ

パラメータ	説明
`parent`	親ウィンドウを指定します。ダイアログがモーダルになります。
`icon`	アイコンの種類を指定します（`error`, `info`, `question`, `warning`）。
`type`	ボタンの種類を指定します（`abortretryignore`, `ok`, `okcancel`, `retrycancel`, `yesno`, `yesnocancel`）。
`default`	デフォルトで選択されるボタンを指定します。

実用的な例

ファイル操作の確認ダイアログ

```
 import tkinter as tk
from tkinter import messagebox, filedialog
import os

class FileManagerApp(tk.Tk):
    def __init__(self):
        super().__init__()

        self.title("ファイルマネージャー")
        self.geometry("400x300")

        self.current_file = None
        self.create_widgets()

    def create_widgets(self):
        # ファイル操作ボタン
        button_frame = tk.Frame(self)
        button_frame.pack(pady=20)

        tk.Button(button_frame, text="新規作成", command=self.new_file, width=12).pack(side=tk.LEFT, padx=5)
        tk.Button(button_frame, text="ファイルを開く", command=self.open_file, width=12).pack(side=tk.LEFT, padx=5)
        tk.Button(button_frame, text="保存", command=self.save_file, width=12).pack(side=tk.LEFT, padx=5)

        # テキストエリア
        self.text_area = tk.Text(self, wrap=tk.WORD, font=("Consolas", 11))
        self.text_area.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # 危険な操作ボタン
        danger_frame = tk.Frame(self)
        danger_frame.pack(pady=10)

        tk.Button(danger_frame, text="全て削除", command=self.clear_all, bg="red", fg="white", width=12).pack(side=tk.LEFT, padx=5)
        tk.Button(danger_frame, text="ファイル削除", command=self.delete_file, bg="darkred", fg="white", width=12).pack(side=tk.LEFT, padx=5)
        tk.Button(danger_frame, text="アプリ終了", command=self.quit_app, width=12).pack(side=tk.LEFT, padx=5)

        # ステータス
        self.status_label = tk.Label(self, text="準備完了", relief=tk.SUNKEN, anchor=tk.W)
        self.status_label.pack(side=tk.BOTTOM, fill=tk.X)

    def new_file(self):
        if self.check_unsaved_changes():
            self.text_area.delete(1.0, tk.END)
            self.current_file = None
            self.status_label.config(text="新しいファイルを作成しました")
            messagebox.showinfo("成功", "新しいファイルを作成しました。")

    def open_file(self):
        if self.check_unsaved_changes():
            filename = filedialog.askopenfilename(
                title="ファイルを開く",
                filetypes=[("テキストファイル", "*.txt"), ("すべてのファイル", "*.*")]
            )
            if filename:
                try:
                    with open(filename, 'r', encoding='utf-8') as file:
                        content = file.read()
                        self.text_area.delete(1.0, tk.END)
                        self.text_area.insert(1.0, content)
                        self.current_file = filename
                        self.status_label.config(text=f"ファイルを開きました: {os.path.basename(filename)}")
                        messagebox.showinfo("成功", f"ファイルを開きました:\n{os.path.basename(filename)}")
                except Exception as e:
                    messagebox.showerror("エラー", f"ファイルを開けませんでした:\n{str(e)}")

    def save_file(self):
        if not self.current_file:
            filename = filedialog.asksaveasfilename(
                title="ファイルを保存",
                defaultextension=".txt",
                filetypes=[("テキストファイル", "*.txt"), ("すべてのファイル", "*.*")]
            )
            if not filename:
```

```
                return
            self.current_file = filename

        try:
            content = self.text_area.get(1.0, tk.END)
            with open(self.current_file, 'w', encoding='utf-8') as file:
                file.write(content)
            self.status_label.config(text=f"ファイルを保存しました: {os.path.basename(self.current_file)}")
            messagebox.showinfo("成功", f"ファイルを保存しました:\n{os.path.basename(self.current_file)}")
        except Exception as e:
            messagebox.showerror("エラー", f"ファイルを保存できませんでした:\n{str(e)}")

    def clear_all(self):
        # 複数段階の確認
        if messagebox.askokcancel("確認", "本当にすべてのテキストを削除しますか？"):
            if messagebox.askyesno("最終確認", "この操作は元に戻せません。\n本当に削除しますか？", icon="warning"):
                self.text_area.delete(1.0, tk.END)
                self.status_label.config(text="すべてのテキストを削除しました")
                messagebox.showinfo("完了", "すべてのテキストを削除しました。")

    def delete_file(self):
        if not self.current_file:
            messagebox.showwarning("警告", "削除するファイルが選択されていません。")
            return

        filename = os.path.basename(self.current_file)

        # 危険な操作の確認
        result = messagebox.askyesnocancel(
            "ファイル削除",
            f"ファイル '{filename}' を完全に削除しますか？\n\nこの操作は元に戻せません。",
            icon="warning"
        )

        if result is True:  # 「はい」が選択された
            try:
                os.remove(self.current_file)
                self.text_area.delete(1.0, tk.END)
                self.current_file = None
                self.status_label.config(text=f"ファイル '{filename}' を削除しました")
                messagebox.showinfo("削除完了", f"ファイル '{filename}' を削除しました。")
            except Exception as e:
                messagebox.showerror("エラー", f"ファイルを削除できませんでした:\n{str(e)}")
        elif result is False:  # 「いいえ」が選択された
            messagebox.showinfo("キャンセル", "ファイル削除をキャンセルしました。")
        # result が None の場合（「キャンセル」）は何もしない

    def quit_app(self):
        if self.check_unsaved_changes():
            if messagebox.askokcancel("終了確認", "アプリケーションを終了しますか？"):
                self.quit()

    def check_unsaved_changes(self):
        # 実際の実装では、変更の有無をチェックする
        content = self.text_area.get(1.0, tk.END)
        if content.strip():  # 何かテキストがある場合
            result = messagebox.askyesnocancel(
                "未保存の変更",
                "変更が保存されていません。\n保存しますか？"
            )

            if result is True:  # 保存する
                self.save_file()
                return True
            elif result is False:  # 保存しない
                return True
            else:  # キャンセル
                return False
        return True

if __name__ == "__main__":
    app = FileManagerApp()
    app.mainloop()
```

ベストプラクティス

プラクティス	説明
適切なダイアログ選択	目的に応じて適切な種類のダイアログを選択します（情報表示、確認、エラー通知など）。
わかりやすいメッセージ	ユーザーが理解しやすい明確で簡潔なメッセージを作成します。
適切なタイトル	ダイアログの目的を明確に示すタイトルを設定します。
重要な操作の確認	削除や終了など、重要な操作には確認ダイアログを表示します。
エラーハンドリング	例外が発生した場合は、適切なエラーメッセージを表示します。

参考リンク

- Python Docs - tkinter.messagebox
- TkDocs - Message Boxes