

JRAーVAN Data Lab. SDK

開発ガイド(イベント Python)

2026年7月31日

第1. 0. 0版

JRAシステムサービス株式会社

修正履歴

日付	版	項番	種類	内容
2026/7/31	1.00	一	新規	初版作成



目次

はじめに

- 1. 1 JV-Linkを使ったプログラミング(イベント取得)
 - 1. 1. 1 プログラミングのゴール
 - 1. 1. 2 プロジェクトの作成
 - 1. 1. 3 JV-Link オブジェクトの取得
 - 1. 1. 4 フォームの作成
 - 1. 1. 5 設定ボタンのコーディング(JVSetUIProperties)
 - 1. 1. 6 初期化／終了処理のコーディング
 - 1. 1. 7 イベント受信処理のコーディング
 - 1. 1. 8 実行



はじめに

平素よりJRA-VANをご利用いただき誠にありがとうございます。

弊社では JRA 公式データをより一層活用できる仕組みとして、「JRA-VAN Data Lab.」サービスを提供しております。本サービスは、データリンクモジュールのJV-Link(※)を通じて、様々な競馬データの利用が可能になるサービスです。

エンドユーザー様、およびソフトウェア作者の皆様には、この「JRA-VAN Data Lab.」サービスの仕組みをご理解いただき、JRA-VANのデータ提供サービスを活用していただきたいと願っております。

※JV-Linkの詳細については JRA-VAN Data Lab.開発ガイドの本編を参照して下さい。

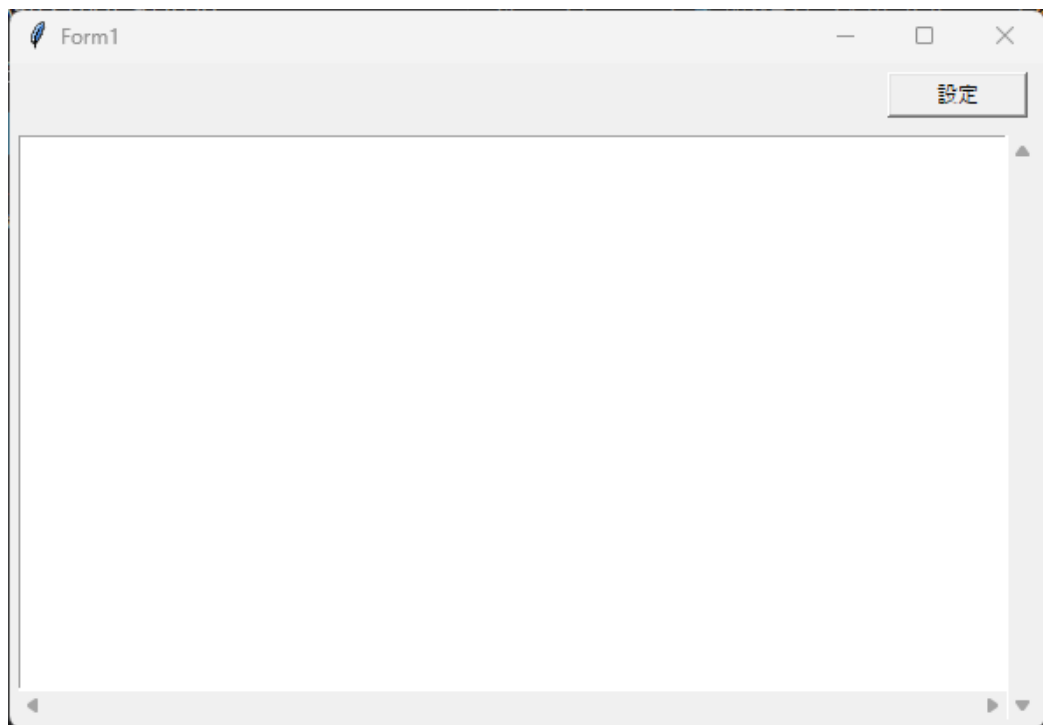
1. 1 JV-Linkを使ったプログラミング(イベント取得)

具体的な例を示すことによって、JV-Linkを使ってイベントを取得する方法について解説していきたいと思います。ここではPythonを例に順を追ってプログラミング方法を説明していきます。JV-Linkのインターフェースの詳細については触れませんので、インターフェースの詳細については「JV-Linkインターフェース仕様書」を参照して下さい。

また、本書ではPythonがインストール済みであることを前提として説明します。あらかじめPythonをインストールしたうえで、本書の手順を実施してください。

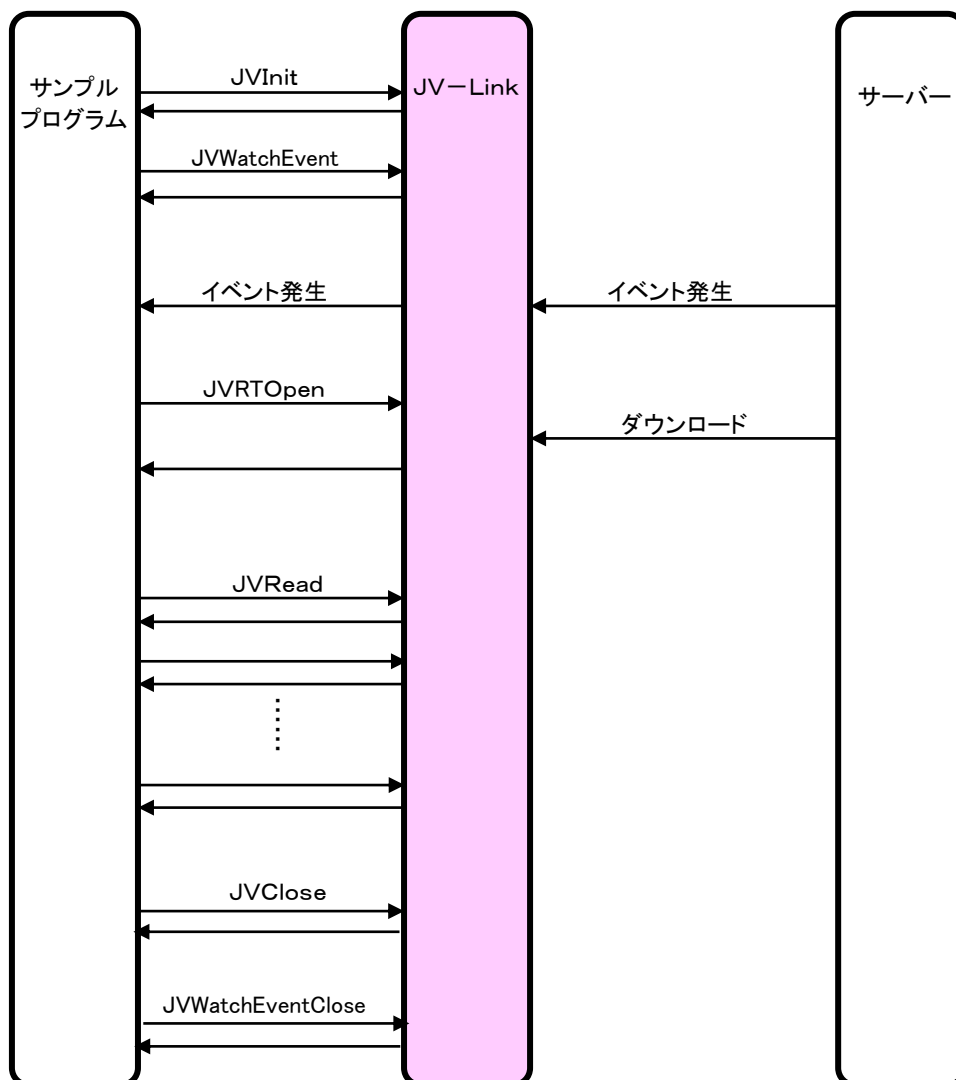
1. 1. 1 プログラミングのゴール

ここではJV-Linkを使ってイベントを取得して表示する簡単なプログラム(Sample2)を作成します。



このプログラムはボタンが1つとテキストボックスが1つのシンプルな画面を持っています。「設定」ボタンを押すとJV-LinkのJVSetUIPropertiesインターフェースを使ってJV-Linkの各種設定を行なうことができます。イベントが発生すると、その内容がテキストボックスに表示されます。

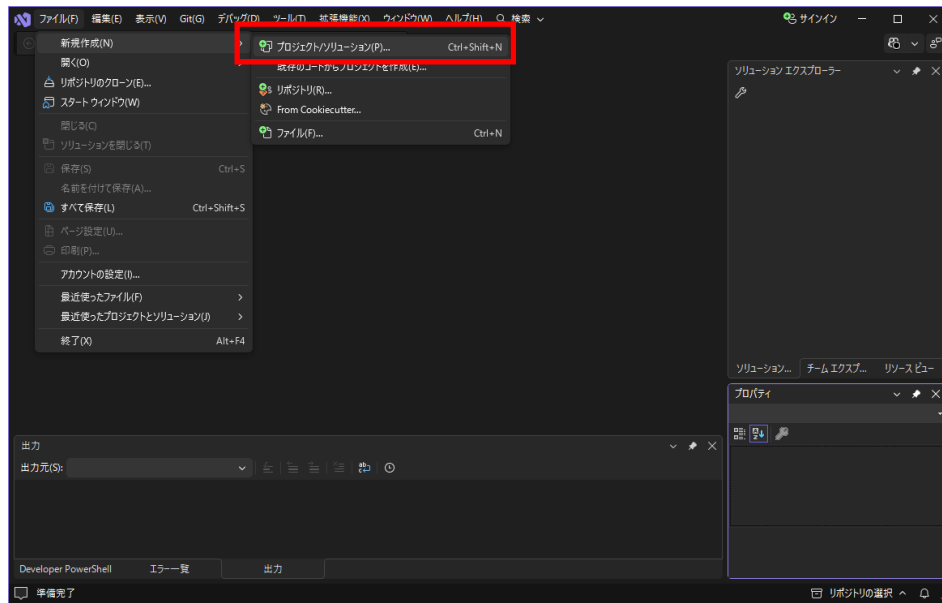
イベント取得で、JV-Link、サーバー間で行なわれる処理は以下のようになります。



1. 1. 2 プロジェクトの作成

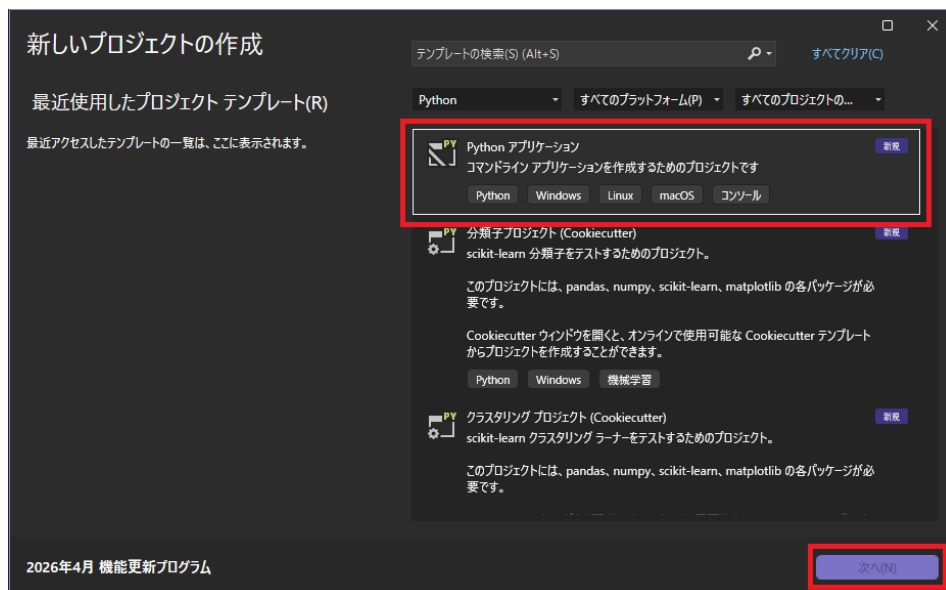
Visual Studio 2026 を起動し、「コードなしで続行 (W)」ボタンをクリックします。

メニューバーのファイル(F)→新規作成(N)→プロジェクト/ソリューション(P) を選択します。

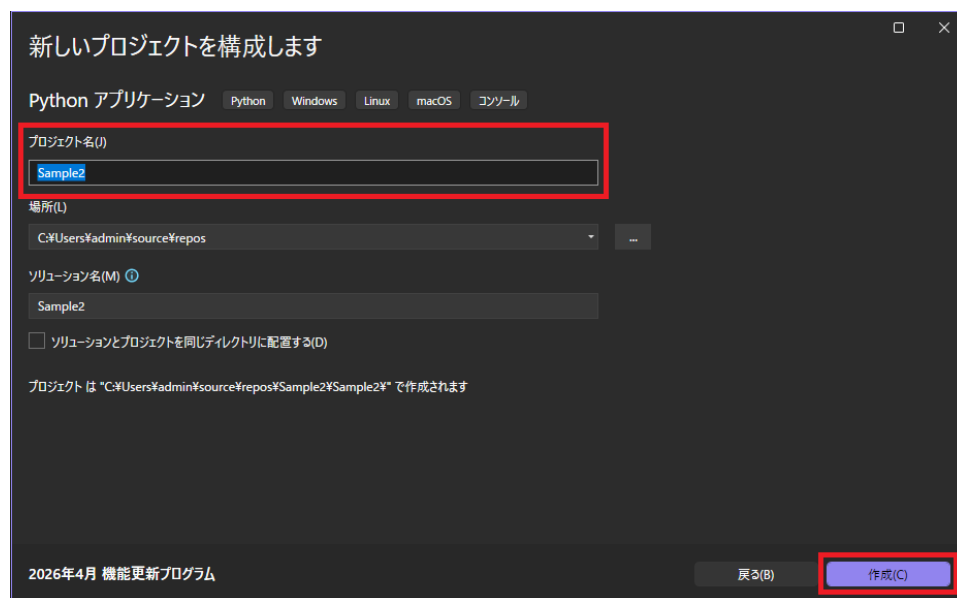


次に、プロジェクトのテンプレートから「Python アプリケーション」を選択して「次へ(N)」ボタンをクリックします。

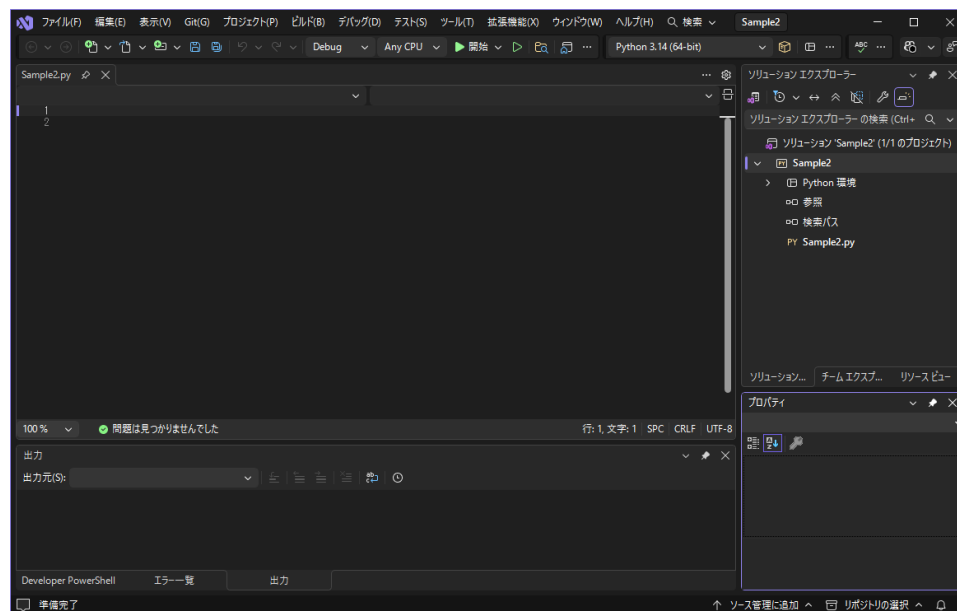
※「Python アプリケーション」が表示されない場合は、Visual Studio Installer にて「Python 開発」ワークロードをインストールしてください。



次に、プロジェクト名(J)を「Sample2」に設定し、その他は任意の設定をして「作成(C)」ボタンをクリックします。



新しいプロジェクトが作成され下のような画面になります。

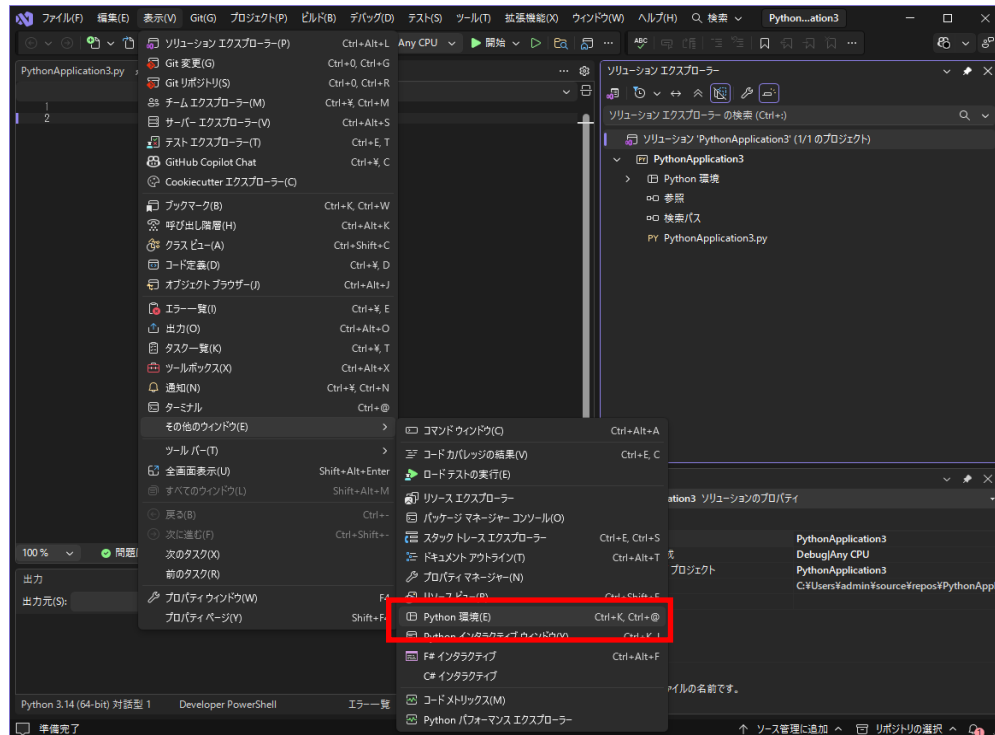


1. 1. 3 JV-Link オブジェクトの取得

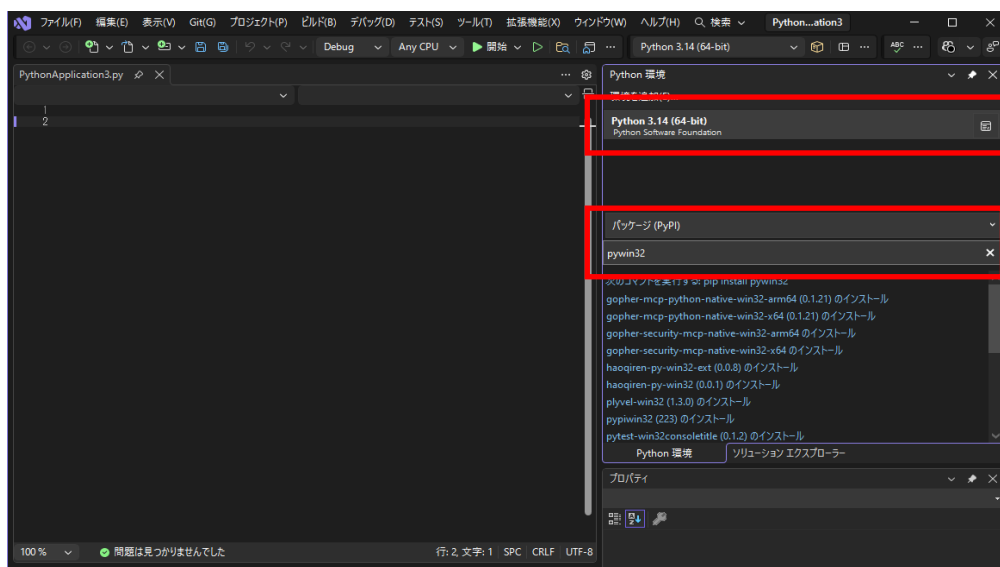
「JV-Link」を Python で使用するには pywin32 で JV-Link オブジェクトを取得する必要があります。

pywin32 はインストールが必要なので、以下手順を実施してください。

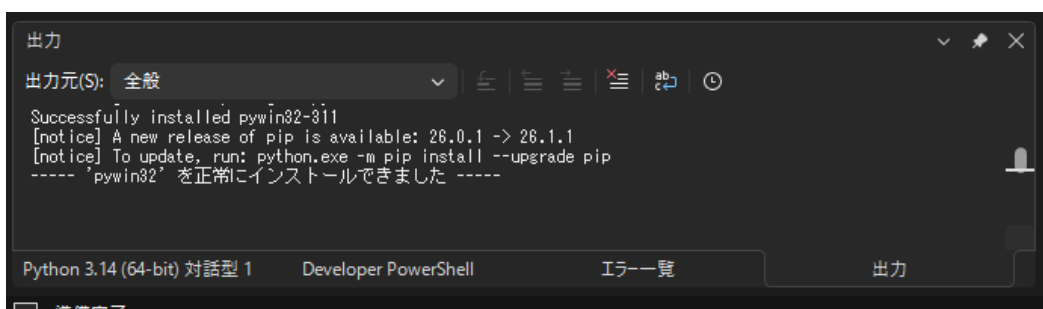
メニューバーの「表示(V)」→「その他のウィンドウ(E)」→「Python 環境(E)」を選択します。



使用している Python のバージョン(画像では Python 3.14)を選択し、パッケージタブを開いて検索欄に「pywin32」と入力します。検索結果の「次のコマンドを実行する: pip install pywin32」または「pywin32 のインストール」をクリックします。



コンソールに以下のように出力されればインストール完了です。



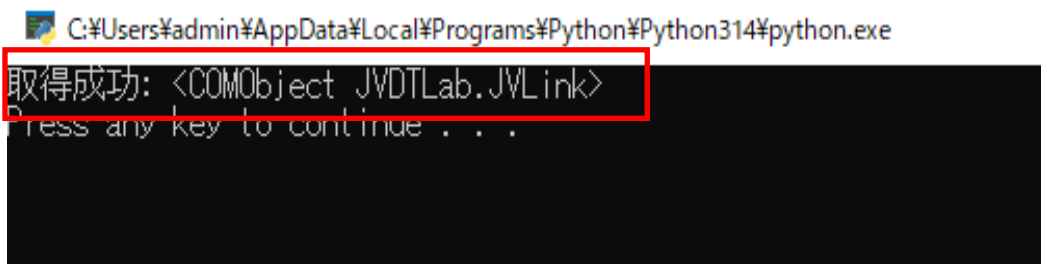
続いて、エディタに以下のコードを記載してください。

```
import win32com.client

# JvLinkオブジェクト取得
jv = win32com.client.Dispatch("JVDTLab.JVLink")

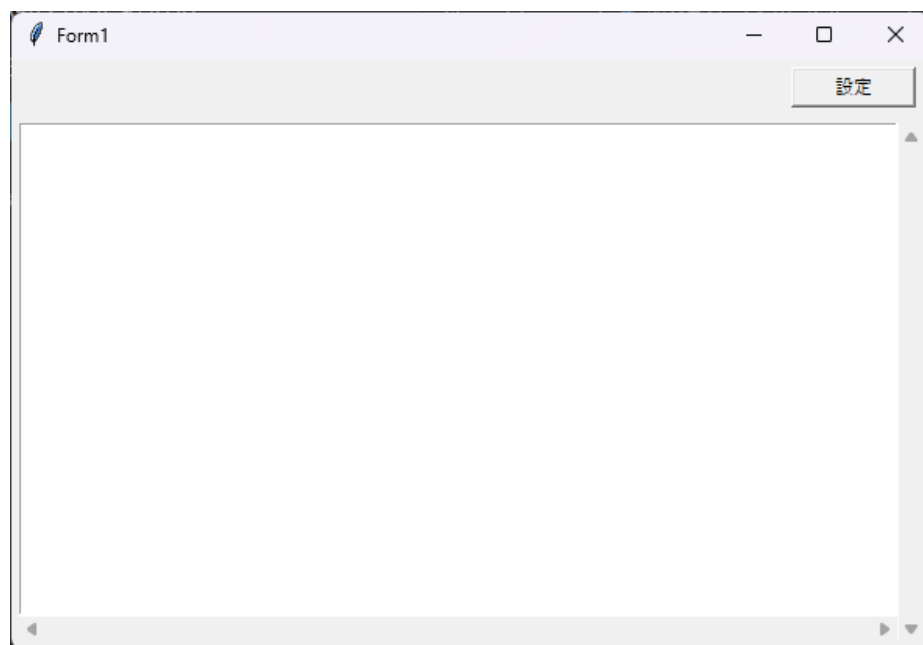
print("取得成功:", jv)
```

上記のコードを実行して以下の内容が出力できれば、JVLink オブジェクトを取得できています。



1. 1. 4 フォームの作成

下図のようなフォームを作成します。Sample2.py にサンプルソース 1 を記載してください。



サンプルソース1

```

# COM操作 (JV-Linkを扱うため)
import win32com.client
import pythoncom

# GUI (Tkinter)
from tkinter import messagebox
import tkinter as tk
from tkinter.scrolledtext import ScrolledText
import threading
from tkinter import ttk

# -----
#   メインGUIクラス
# -----
class App:
    def __init__(self, root):
        self.root = root
        self.root.title("Form1")

        # -----
        #   上部フレーム (ボタン)
        # -----
        top_frame = tk.Frame(root)
        top_frame.pack(fill=tk.X, padx=5, pady=5)

        self.setting_button = tk.Button(
            top_frame,
            text="設定",
            width=10,
            command=self._on_setting
        )
        self.setting_button.pack(side=tk.RIGHT, padx=5)

        # -----
        #   メインフレーム (テキスト表示)
        # -----
        frame = tk.Frame(root)
        frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        # 縦スクロールバー
        y_scroll = ttk.Scrollbar(frame, orient="vertical")
        y_scroll.pack(side=tk.RIGHT, fill=tk.Y)

        # 横スクロールバー
        x_scroll = ttk.Scrollbar(frame, orient="horizontal")
        x_scroll.pack(side=tk.BOTTOM, fill=tk.X)

        # テキストエリア
        self.text = tk.Text(frame, wrap="none")
        self.text.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

```

```

# スクロール連動設定
self.text.configure(
    xscrollcommand=x_scroll.set,
    yscrollcommand=y_scroll.set
)

x_scroll.config(command=self.text.xview)
y_scroll.config(command=self.text.yview)

# -----
# JVLink関連
# -----
self.jvlink = None # 初期はNone
self.running = True

# JVLinkを別スレッドで起動
threading.Thread(target=self.run_jvlink, daemon=True).start()

# ウィンドウ閉じるイベント
self.root.protocol("WM_DELETE_WINDOW", self.on_close)

# -----
# テキスト出力
# -----
def append_text(self, msg):
    self.text.insert(tk.END, msg)
    self.text.see(tk.END)

# -----
# ウィンドウ終了
# -----
def on_close(self):
    self.running = False
    self.root.destroy()

# -----
# メイン処理
# -----
def main():
    root = tk.Tk()
    App(root)
    root.mainloop()

if __name__ == "__main__":
    main()

```

1. 1. 5 設定ボタンのコーディング (JVSetUIProperties)

Sample2 では設定ボタンをクリックすることによって **JVSetUIProperties** インターフェースを呼び出し、JVLinkコントロールが持つ設定ダイアログを起動します。

サンプルソース1のメイン処理の上に以下のコードを記述してください。

サンプルソース2

```
# -----
# 設定ボタン押下時の処理
# -----
def _on_setting(self):
    # 設定画面表示
    if self.jvlink.JVSetUIProperties() == -100:
        messagebox.showerror("エラー", "JV-Link の設定に失敗しました")
    return
```

サンプルソース2解説

①JVSetUIProperties を呼び出した結果が-100 の場合は設定が何らかのエラーでレジストリに反映できなかったことを示します。

1. 1. 6 初期化／終了処理のコーディング

初期化処理として以下の処理を、ウィンドウ起動時に行います。

処理	内容
①JVLinkの初期化 (JVInit)	JVLinkの初期化を行います。JVWatchEvent を呼び出す前に最低1回呼び出す必要があります。
②イベント通知開始 (JVWatchEvent)	確定・変更情報が発生した際、イベントを通知するスレッドを開始します。

終了処理として以下の処理を、ウィンドウクローズイベントで行います。

処理	内容
①イベント通知終了 (JVWatchEventClose)	イベント通知スレッドを終了します。

以下のコードを、サンプルソース1のメイン処理の上に記述してください。

サンプルソース3

```
#-----
#   JVLink初期化スレッド
#-----
def run_jvlink(self):
    # COM開始・初期化
    pythoncom.CoInitialize()

    # JVLink取得 + イベントバインド
    self.jvlink =
        win32com.client.DispatchWithEvents("JVDTLab.JVLink", JVLinkEventHandler) ①

    # イベント内からJVLinkを使えるように設定
    self.jvlink.jvlink = self.jvlink

    # イベント内からGUIを使えるように設定
    self.jvlink.set_gui(self)

    # -----
    # 初期化
    # -----
    ret = self.jvlink.JVInit("UNKNOWN") ②
    if ret != 0:
        self.append_text(f"JVInitエラー: {ret}¥n")
        return

    # -----
    # イベント監視開始
    # -----
    ret = self.jvlink.JVWatchEvent() ③
    if ret != 0:
        self.append_text(f"JVWatchEventエラー: {ret}¥n")
        return

    # イベント待機ループ
    while self.running:
        pythoncom.PumpWaitingMessages()

    # -----
    # 終了処理
    # -----
    self.jvlink.JVWatchEventClose() ④
    self.append_text("JVWatchClose完了¥n")

    # COM終了
    pythoncom.CoUninitialize()
```

サンプルソース3解説

- ①JVLink の取得と JV-Link から発生するイベントを JVLinkEventHandler クラスで受け取れるようにしています。
“JVLinkEventHandler”の部分は、イベント受信時の処理を記述するクラス名にしてください。
- ②JVInitにはアプリケーションIDを渡します。このアプリケーションIDはサーバーとの通信の際にHTTPヘッダーのUser-Agentとして使用されます。ソフトの名前とバージョン番号を組み合わせるユニークなIDを指定して下さい。開発途中でアプリケーションIDを決められない場合には“UNKNOWN”を指定して下さい。
- ③イベント通知開始をおこなうことで、それ以降に払戻確定、騎手変更、天候馬場状態変更、コース変更、出走取消・競走除外、発走時刻変更、馬体重が発表された際、イベントを受信することが可能になります。
- ④イベント受信処理を終了する場合に JVWatchEventClose を呼び出します。今回のサンプルではフォームを閉じる時(プログラムが終了する時)に呼び出しています。

1. 1. 7 イベント受信処理のコーディング

各イベントに対してのイベント関数を作成することにより、対象イベントが発生した際の処理をコーディングすることが可能となります。

ここではサンプルとして、払戻確定イベントが発生した際のコードを載せますが、それ以外のイベントも作り方は同じです。

以下のコードをサンプルソース1のメイン処理の上に記述し、さらに読み込み&表示処理の関数を作成します。

このサンプルでは、イベントが発生した元となるデータを取得し画面に表示しています。

処理	内容
①リアルタイム系データの取得要求(JVRTOpen)	リアルタイム系データの取得要求をします。
②JV-Data の読み込み(JVRead)	JV-Data の読み込みをします。
③JV-Data 読み込み処理の終了(JVClose)	JV-Data 読み込み処理の終了をします。

サンプルソース4

```

#-----
# イベントハンドラクラス
#-----
class JVLinkEventHandler:
    def set_gui(self, gui):
        self.gui = gui

#-----
# 払戻速報
#-----

    def OnJVEvtPay(self, bstr):
        # GUIに表示
        self.gui.append_text(f"払戻速報: {bstr}\n")
        # 払戻データ取得の開始
        return_code = self.jvlink.JVRTOpen("0B12", bstr)
        if return_code != 0:
            # エラー
            self.gui.append_text(f"JVRTOpenエラー: {return_code}\n")
        else:
            # 正常
            self.gui.append_text(f"【速報レース情報(払戻速報) dataspec=0B12 key={bstr}】\n")
            # 読み込み&表示処理
            self.read_data()

#-----
# 読み込み&表示処理
#-----

    def read_data(self):
        # JV-Linkからデータ受け取るバッファ
        buff = bytearray()
        fname = bytearray()
        buff_size = 150000

        while True:
            # 1レコード取得
            return_code, memview, fname = self.jvlink.JVGets(buff, buff_size, fname)

            if return_code > 0:
                # データあり
                raw: bytes = memview.tobytes()

                # 文字コード変換 (JV-Linkはcp932)
                text = raw.decode("cp932", errors="replace")

                # GUI表示
                self.gui.append_text(text)

            elif return_code == 0:
                # 読み込み終了
                break
            elif return_code == -1:
                pass
            else:
                # エラー
                self.gui.append_text(f"JVGetsエラー: {return_code}\n")
                break

        # 読み込み終了
        self.jvlink.JVClose()

```

サンプルソース4解説

①イベント発生元となったリアルタイム系データの取得要求をします。

第1パラメータはイベント毎に決まっている「データ種別ID」です。

詳細については「JV-Linkインターフェース仕様書」を参照して下さい。

②JVROpen で準備した JV-Data を現在のファイルポインタから1行分読み出します。

正の数が返された場合は、正常にレコードを読み込んでいるので(文末に改行コード付)データ格納バッファを画面にそのまま表示します。

-1が返された場合は、読込対象ファイルの変更を意味しているので、何もせず次のデータを読み込みにいきます。

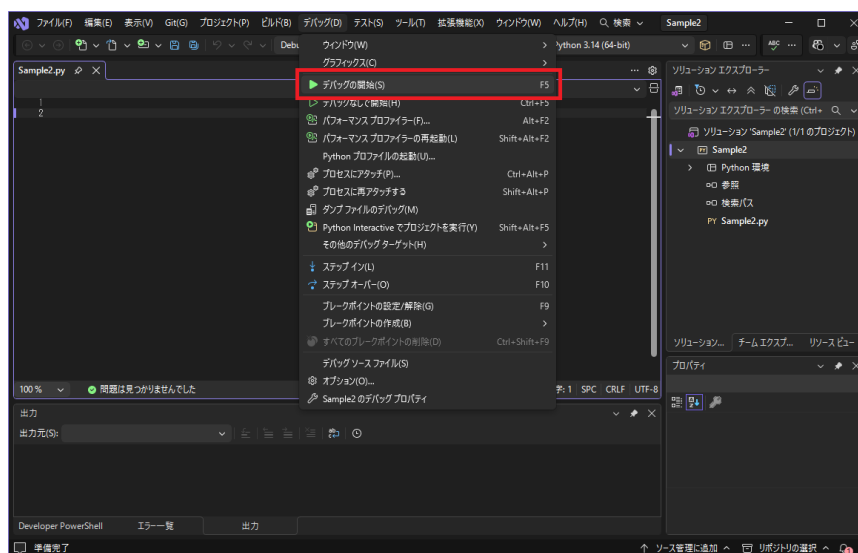
全てのファイルを読み終わると0が返りますので、読み込みループを終了します。エラーが発生した場合にはエラーの理由コードとして負の数が返されます。JVGets ではなく JVRead を使って読み出すことも可能です。

③JV-Data読み込み処理を終了します。

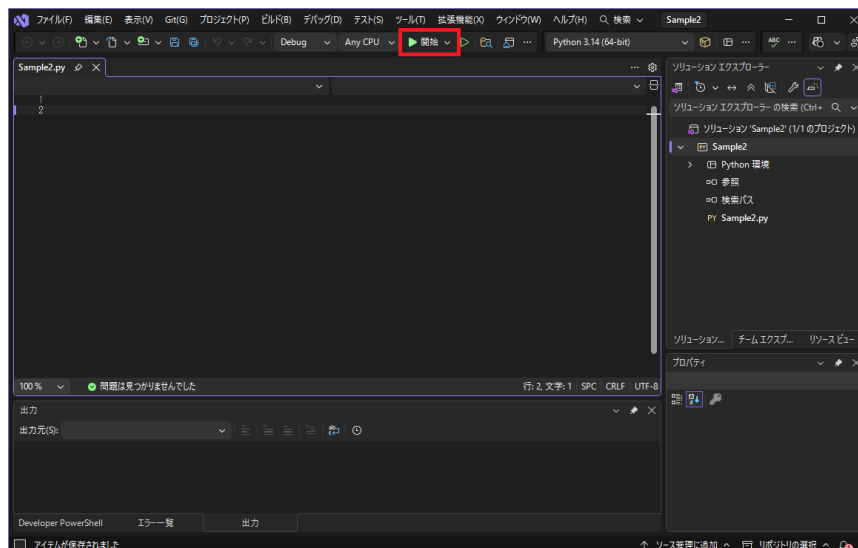
1.1.8 実行

以上で Sample2 の開発は終了です。以下の手順で実行が可能です。

メニューバーのデバッグ(D)→デバッグの開始(S)を選択します。



または、ツールバーの「開始」ボタンをクリックします。

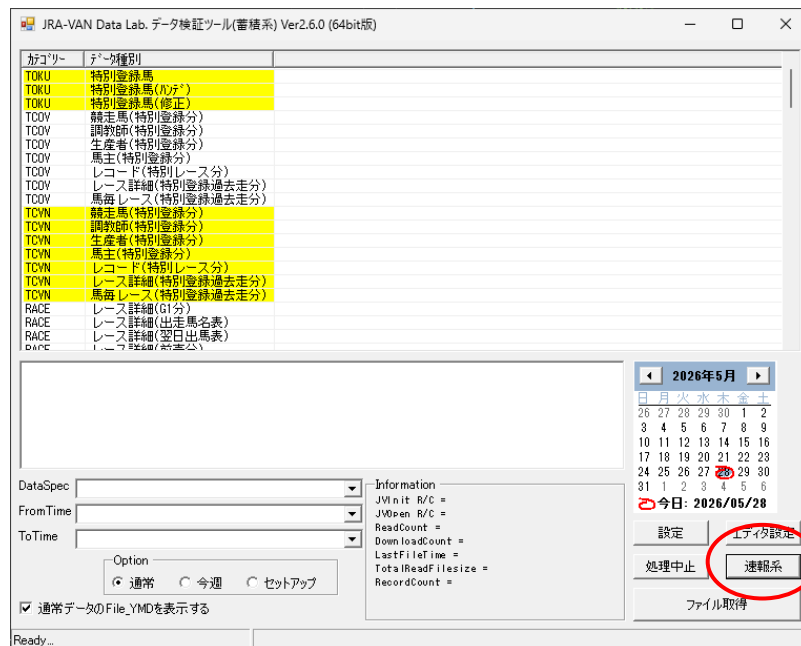


実際にイベントを発生させて、その動作を検証するには DataLab.検証ツールを使います。
 (DataLab.検証ツールはJRA-VAN のホームページの「プログラミングパーツ・開発支援ツール
 提供コーナー」にありますので、ダウンロードしてください。)

DataLab.検証ツールのインストール及び詳しい使用方法に関しては、DataLab.検証ツール説明書を参照してください。

今回サンプルソースで作成した、払戻確定イベントを発生させるには以下の様に操作します。

- ① DataLab.検証ツールを起動します。
- ② 「速報系」ボタンをクリックします。



- ④ 拡張結果が Sample2 に表示されます。
- DataLab.検証ツールと同じデータが表示されていることを確認してください。