VPF プラグイン作成方法

第1版

Contents

1. VPFで扱うプラグインについて

本書は、VPFで使用可能なプラグインを作成する手順について説明したものです。

VPFでは、プラグイン開発用の基底クラスとしてPluginBaseクラスを定義しており、
このクラスを継承して作成したsoファイルのみをプラグインとして扱います。

以降の章では、独自プラグインを作成する上で、
カスタマイズ必須となるPluginBaseクラスのメンバ及び、
当該メンバ内で必要な実装に関する記載を行います。

2. プラグイン作成概要

ここでは、プラグインを作成するための概要を記載します。
PluginBaseで提供しているメソッドには、**カスタマイズ必須な処理**と、
必要な場合のみ実装が必要な処理が存在します。
以降の章では、必須な処理から順に作成方法を記載します。

PluginBaseクラスメソッド名	概要	実装必要な処理
コンストラクタ	-	<<必須>> ・プラグイン名設定 ・入出力ポート仕様設定 ・出力バッファ設定
InitProcess(CommonParam* common)	VPF再生時の初期化処理	<<Optional>> ・CommonParameter設定
DoProcess(cv::Mat* src_image, cv::Mat* dst_image)	VPF再生中の画像処理等	<<必須>> ・画像処理
OpenSettingWindow(ImageProcessingState state)	VPF設定画面表示	<<Optional>> ・設定画面
set_image_processing_state(ImageProcessingState state)	VPF実行状態通知	<<Optional>> ・設定画面状態変更
ClearPluginSettings(void) AddLinePluginSettings(wxString params) SetPluginSettings(std::vector<wxString> params)	フロー保存機能の Save/Loadにプラグイン設定 値保存組み込み用	<<Optional>> ・設定値保存/読み込み処理

3. カスタマイズ必須処理

ここでは、PluginBaseを継承したクラスで
カスタマイズ必須となる下記メソッドについて記載します。

- 3-1 コンストラクタ
- 3-2 DoProcess

3-1. <<必須>> コンストラクタ

コンストラクタは、VPF本体側でプラグイン読み込み時に実行され、
下記①～③を実装する必要があります。
①プラグイン名設定
VPFで扱うプラグイン名の設定で、任意の名前を指定する必要があります。
<<プラグイン名設定>>
void set_plugin_name(const std::string plugin_name)
- plugin_name : プラグイン名

補足）
未設定の場合、”Unknown”と表示されます。

3-1. <<必須>> コンストラクタ（続き）

②ポート仕様設定
VPFでは各プラグインの入力及び出力ポート設定を事前にチェックし、接続可能なプラグイン同士のみを繋ぐ仕組みを用意しています。
（ポートに関して、詳細な定義は01_VPF機能仕様書.pptx内、3-2.1を参照）

<<入力ポート仕様設定(※1)>>
int AddInputPortCandidateSpec(PlaneType type)
- type：入力ポート種別
- 戻り値：Index(※2)
<<出力ポート仕様設定(※1)>>
int AddOutputPortCandidateSpec(PlaneType type)
- type：出力ポート種別
- 戻り値：Index(※2)
<<入出力ポートの関連性を設定(※3)>>
bool AddPortRelation(unsigned int input_index, unsigned int output_index)
- input_index：入力ポートIndex(※2)
- output_index：出力ポートIndex(※2)
- 戻り値：関連性設定結果(true = 使用可能ポート, false = 使用不可ポート)

※1 入力または出力のどちらかを必要としないプラグインの場合、使用しないポートの設定を行う必要はありません。
8bit/10bit向け等、複数ポートをサポートしている分は、使用するポート数分ポート設定を行う必要があります。
※2 AddInputPortCandidateSpec(), AddOutputPortCandidateSpec()実行時、一意なIndexを戻り値として返却します。
その値は、AddPortRelation()で使用します。
※3 入力または出力のどちらかを必要としないプラグインの場合、AddPortRelation()を実行する必要はありません。

3-1. <<必須>> コンストラクタ（続き）

③出力バッファ設定
VPFでは、下記2通りのプラグイン実行方式をサポートしており、
プラグインの実装に応じてどちらのImageData領域を使うか指定する必要があります。

1.前段プラグインから
　　通知されたImageData領域上で、直接画像処理を行う
2.前段プラグインから
　　通知されたImageDataと異なる領域を使用し、画像処理を行う
　（ImageData領域に関して、詳細な定義は3-2を参照）

<<出力バッファ設定>>
void set_is_use_dest_buffer(bool is_use_dest)
- is_use_dest：false = 上記1の設定
true = 上記2の設定

補足）
未設定の場合、上記2の設定が適用されます。

3-1. <<必須>> コンストラクタ（続き）

以下にコンストラクタの実装例を記載します。

```
Template::Template() : PluginBase() {

  // Initialize base class(plugin_base.h)
  set_plugin_name("Template");

  // Create input port.
  int input_port_id_1 = AddInputPortCandidateSpec(kGRAY8);
  int input_port_id_2 = AddInputPortCandidateSpec(kGRAY16);

  // Create output port.
  int output_port_id_1 = AddOutputPortCandidateSpec(kGRAY8);
  int output_port_id_2 = AddOutputPortCandidateSpec(kGRAY16);

  // Create port relation.
  bool is_connect_relation_1 = AddPortRelation(input_port_id_1, output_port_id_1);
  bool is_connect_relation_2 = AddPortRelation(input_port_id_2, output_port_id_2);

  // Check port relation.
  if (is_connect_relation_1 == false || is_connect_relation_2 == false) {
    DEBUG_PRINT("Template port relation fail¥n");
    is_success_port_relation_ = false;
  } else {
    is_success_port_relation_ = true;
  }

  // Use only src buffer
  set_is_use_dest_buffer(true);
}
```

① .任意のプラグイン名を設定

②-1.入力ポートを設定
- サポートしているポート数分記載必要
- 入力ポートが不要なプラグインでは記載不要

②-2.出力ポートを設定
- サポートしているポート数分記載必要
- 出力ポートが不要なプラグインでは記載不要

②-3.入出力ポートの関連性を設定
- 一方のポートのみ使用するプラグインでは記載不要

②-4.関連性設定結果チェック
有効な関連性がない場合、
VPFでは当該プラグインを動作不可なものと判断します。
この場合、VPFの再生を実行しても直後に停止します。

③ .出力バッファを設定

3-2. <<必須>> DoProcess

DoProcessは、VPF本体側で再生操作を行った場合に実行される画像処理用のメソッドです。
プラグインに画像処理を組み込む場合、DoProcessをOverrideし、その中に処理を組み込む必要があります。

<<DoProcessフォーマット>>

DoProcess(cv::Mat* src_image, cv::Mat* dst_image)

- ***src_image：前段プラグイン処理結果が格納されたImageData領域***
- ***dst_image：当該プラグインの処理結果格納先のImageData領域(※1)***

※1 3-1章内、①のset_is_use_dest_buffer設定でfalseを指定した場合は、src_imageと同じアドレスが設定されます。

補足）

- 本体側でImageDataの領域を事前に確保します。このため、プラグイン内でsrc_image/dst_imageの領域を確保する必要はありません。
- DoProcess実装例については、各プラグインを御参照下さい。

4. Optionでカスタマイズ必要な処理

ここでは、PluginBaseを継承したクラスのうち、
Optionでカスタマイズ必要な処理について記載します。

- 4-1 InitProcess(CommonParam* common)
- 4-2 OpenSettingWindow(ImageProcessingState state)
- 4-3 set_image_processing_state(ImageProcessingState state)
- 4-4 ClearPluginSettings(void)
- 4-5 AddLinePluginSettings(wxString params)
- 4-6 SetPluginSettings(std::vector<wxString> params)

4-1. <<Optional>> InitProcess

InitProcessは、VPF本体側で再生操作を行った直後に1度のみ実行される初期化用の処理です。
プラグイン実行時に必要な初期化処理があれば、このクラスをOverrideする必要があります。

また、引数として通知されるcommonは、プラグイン間のデータ共有用に提供しているCommon APIにアクセスするためのアドレスです。
このアドレスをプラグイン側で保持しておくことで、再生中にプラグイン間のデータ共有を可能とします。

<<設定画面表示>>
bool InitProcess(CommonParam* common)
- common：プラグイン間データ共有用の情報

4-1. <<Optional>> InitProcess (続き)

Version1.1.1のVPFでは以下の機能をCommon APIとして
サポートしています。

Common API名	概要
void set_first_pixel(int first_pixel)	First pixel値の設定
int first_pixel(void)	First pixel値の取得
void set_optical_black(int optical_black)	Optical black値の設定
int optical_black(void)	Optical black値の取得
void SetOnepushRectabgle(int start_x, int start_y, int end_x, int end_y)	Onepush用矩形の設定
CvRect GetOnepushRectangle(void);	Onepush用矩形の取得

4-2. <<Optional>> OpenSettingWindow

OpenSettingWindowは、VPF本体側で当該プラグインの設定画面表示操作を行った際に実行されるメソッドです。
プラグインに設定画面を持たせるには、OpenSettingWindowをOverrideする必要があります。

<<設定画面表示>>
void OpenSettingWindow(ImageProcessingState state)
- state：VPF実行状態

補足）
・ImageProcessingState：VPF実行状態『kStop(停止中), kRun(実行中), kPause(一時停止中)』を通知します。
VPF実行状態に合わせてメニュー表示状態を変えたい場合等は、
このStateを元に表示状態を変更する必要があります。
・ OpenSettingWindow実装例については、各プラグインを御参照下さい。

4-3. <<Optional>> set_image_processing_state

set_image_processing_stateは、VPF本体側の実行状態変更時に実行されるメソッドです。
VPF有効状態に合わせ、設定画面等の有効状態を変更したい場合、
set_image_processing_stateをOverrideする必要があります。

<<設定画面表示>>
void set_image_processing_state(ImageProcessingState state)
- state：VPF実行状態

補足）
・ImageProcessingState：VPF実行状態『kStop(停止中), kRun(実行中), kPause(一時停止中)』を通知します。
VPF実行状態に合わせてメニュー表示状態を変えたい場合等は、
このStateを元に表示状態を変更する必要があります。
・set_image_processing_state実装例については、各プラグインを御参照下さい。

4-4. <<Optional>> ClearPluginSettings

ClearPluginSettingsは、基底クラスとなるPluginBaseで保持している『画像処理フロー向けプラグイン設定情報領域』をクリアするメソッドです。
フロー保存機能でプラグインのパラメータ値を.flowファイルに保存する場合に使用します。

プラグイン設定情報更新時、本メソッドを呼び出し領域をクリアした後に、
4-5. AddLinePluginSettings()を呼び出し、設定情報を更新して下さい。

<<画像処理フロー向けプラグイン設定情報領域クリア>>
void ClearPluginSettings(void)

補足）
・ ClearPluginSettings使用例については、各プラグインを御参照下さい。

4-5. <<Optional>> AddLinePluginSettings

AddLinePluginSettingsは、基底クラスとなるPluginBaseで保持している『画像処理フロー向けプラグイン設定情報領域』へ行単位で設定情報を書き込むメソッドです。フロー保存機能でプラグインのパラメータ値を.flowファイルに保存する場合に使用します。

プラグイン設定情報更新時は、4-4. ClearPluginSettings()を呼び出した後に、本メソッドを呼び出し設定情報を更新して下さい。

<<画像処理フロー向けプラグイン設定情報追加>>
void AddLinePluginSettings (wxString params)
- params：設定保存対象の文字列

補足）
・ AddLinePluginSettings使用例については、各プラグインを御参照下さい。

4-6. <<Optional>> SetPluginSettings

SetPluginSettingsは、フロー保存機能でプラグインのパラメータ値を.flow
ファイルから読み込んだ際に、本体側からプラグインに対してパラメータ値
を通知するメソッドです。

<<設定画面表示>>
void SetPluginSettings(std::vector<wxString> params)
- ***params：設定情報***

補足）
・SetPluginSettings使用例については、各プラグインを御参照下さい。

Appendix. コーディング規約

コーディング規約はGoogle C++スタイルガイドに基づいています。
スタイルガイド及び、補足事項として別途定めたルールについては、
同梱の下記資料を参照下さい。

03_プラグイン作成方法
- Google C++スタイルガイド 日本語訳.mht
- VisionProcessingFrameworkコーディングルール.docx

出典）
Google C++スタイルガイド 日本語訳
http://www.textdrop.net/google-styleguide-ja/cppguide.xml