Nx Witness Metadata SDK開発ガイド v1.0

Nov, 2024

amnimo™

はじめに

- 本ガイドでは、Nx Witness(以後Nxと略記)に映像分析機能を追加するための Metadata SDKの概要について解説します
  - 本ガイドはNetwork Optix社が公開している情報もとにをアムニモが独自で作成したものであり、内容についてはあくまで参考情報としてご参照ください
    - 一部Metadata SDK(バージョン 5.1.1.37512)内の情報を引用しています
  - より詳細な情報や最新情報については、metadata SDK内のドキュメントや、Network Optix社のWebサイトで公開されているドキュメントを参照ください
  - 本ガイドの内容はアムニモ製AG/AXシリーズのエッジゲートウェイ向けの開発のみを意図しています

目次

- Metadata SDKの概要
- Metadata SDKにより拡張できる機能の内容
- 分析プラグインの概要・クラス構成
- 処理シーケンス概要
- サンプルコード
- Pluginクラスの構成
- Engineクラスの構成
- DeviceAgentクラスの構成
  - DeviceAgentの設定
  - カメラ映像の取得
  - メタデータ(オブジェクト・イベント)の登録
  - 診断イベントの登録
- 実装のベストプラクティス・注意点
- その他TIPSなど
- 公開ドキュメント

Metadata SDKの概要

- Nxの開発元であるNetwork Optix社が公開しているSDKであり、分析プラグインを開発・インストールすることで、Nxに映像分析機能を統合することが可能です
  - Metadata SDKはC++言語で記述されています

- metadata SDKはNetwork Optix社のWebサイトからダウンロードできます
  - https://beta.networkoptix.com/beta-builds/default/index.html

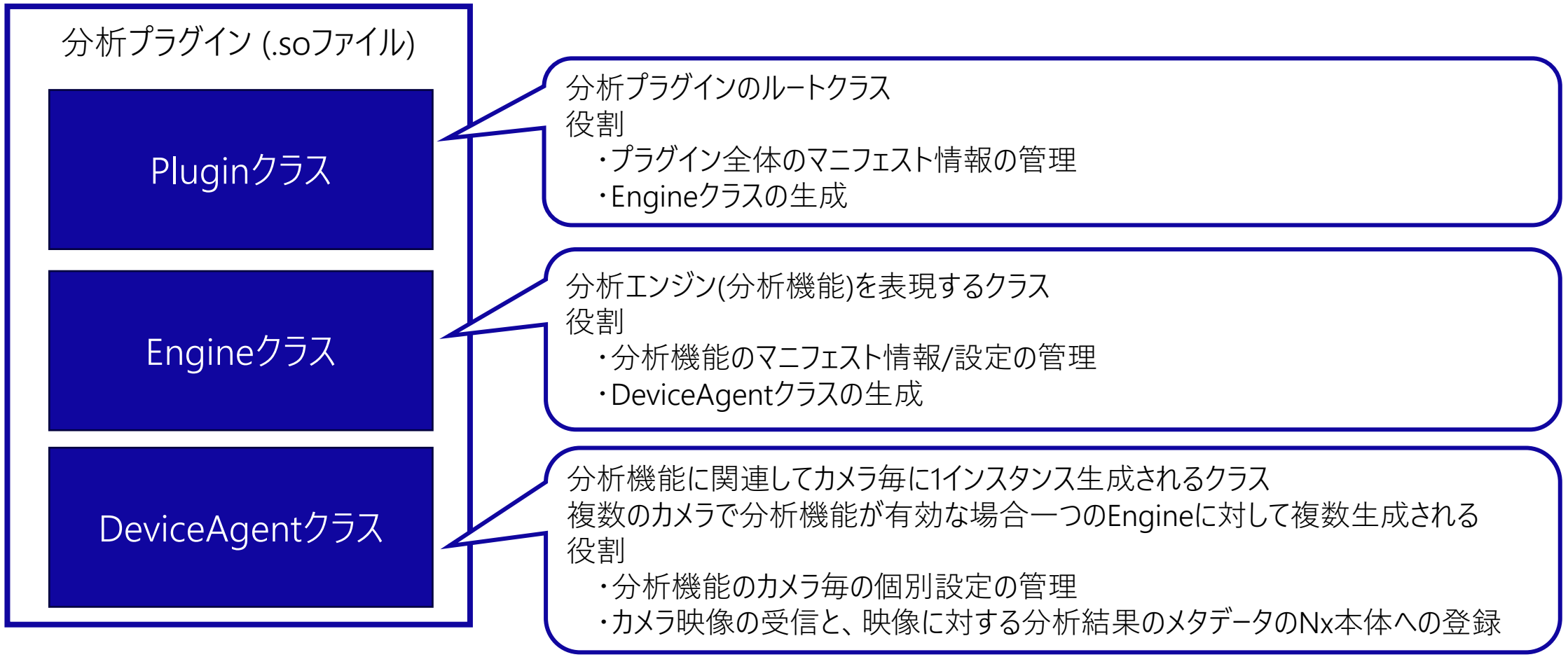
Metadata SDKにより拡張できる機能の内容

- 映像に関連したメタデータの登録
  - メタデータは録画映像と関連付けられてNx内のデータベースに登録されます
  - 以下の種別のメタデータを取り扱うことができます
    - 映像分析によって得られたオブジェクト (映像中の注目する物体)
      - オブジェクトの情報は映像への重畳表示が可能です
    - 拡張機能に関わる設定情報 (例 : 侵入禁止エリアの座標情報)
      - 設定情報のうち、エリアやラインなど一部の情報は映像への重畳表示が可能です
    - 映像に関連するイベント (例 : 侵入禁止エリアへの侵入)
      - このイベントはNx内で分析イベントとして取り扱われ、他のイベントと同様にイベントルールの設定が可能です
    - プラグイン自体の診断イベント (例 : プラグイン処理時のエラー通知など)
- コンテキストアクションの設定
  - 検出した物体に対する操作　(例: 検出した人物へ名称を付与する)

参考URL: Integration Capabilities – What to Expect

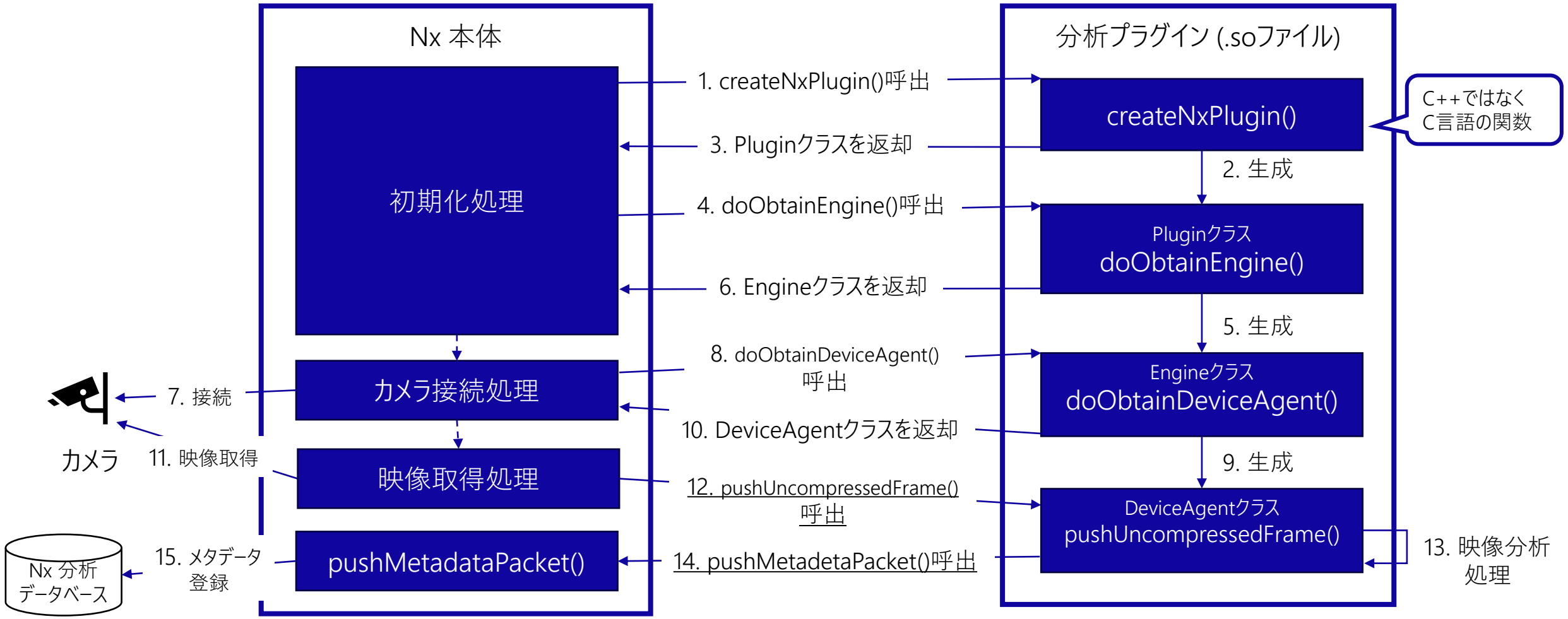
分析プラグインの概要・クラス構成

- 分析プラグインは共有ライブラリとしてビルドされ、所定のディレクトリに配置されます
  - 配置先ディレクトリ : /opt/networkoptix/mediaserver/bin/plugins



処理シーケンス概要

- 下記のシーケンスにおいて、処理12で実際の映像がプラグインに渡され、処理14で分析結果のメタデータをNx本体に登録します
  - 映像を取得する度に処理11から処理15が繰り返し実行されます



参考URL: Plugin Control Flow

サンプルコード

- SDKの samples/以下に2種類のプラグインのサンプルコードが収録されています
  - sample_analytics_plugin/
    - プラグインが成立する最低限のサンプルコード
  - stub_analytics_plugin/
    - Metadata SDKで実現できる各種機能のサンプルコード
- ビルド方法
  - ビルド用のパッケージをインストールする(要インターネット接続)
    - sudo apt install cmake build-essential
  - metadata_sdk/build_samples.sh の 53行目の
    BUILD_OPTIONS=(-- -j) を BUILD_OPTIONS=(-- -j 2) と変更する(" 2"を追記する)
  - build_samples_arm64.sh を実行する
  - ビルド生成物は、Metadata SDKを配置したディレクトリと並列に作成される "metadata_sdk-build"ディレクトリに格納されます

Pluginクラスの構成

- 下記コード例はMetadata SDK内のサンプルコードから引用しています
  - 出展：samples/sample_analytics_plugin/src/nx/vms_server_plugins/analytics/sample/plugin.cpp

関数名	役割	コード例

Plugin::doObtainEngine()	プラグインに対応するEngineクラスのインスタンスを返却します	`Result<IEngine*> Plugin::doObtainEngine()`
		`{`
		`    return new Engine();`
		`}`

Plugin::manifestString()	本プラグインのマニフェスト情報をJSON形式で返却します
	記載内容はSDK内の
	src/nx/sdk/analytics/manifests.md
	Plugin Manifestの章を参照ください
		`std::string Plugin::manifestString() const`
		`{`
		`    return /*suppress newline*/ 1 + (const char*) R"json(`
		`{`
		`    "id": "nx.sample",`
		`    "name": "Sample",`
		`    "description": "A simple \"Hello, world!\" plugin that can be used as a template.",`
		`    "version": "1.0.0",`
		`    "vendor": "Plugin vendor"`
		`}`
		`)json";`
		`}`

createNxPlugin()
C++ではなく
C言語の関数
	共有ライブラリ内に実装されているPluginクラスのインスタンスを返却します
	
	共有ライブラリ内にに複数のプラグインを実装する際はcreateNxPluginByIndex() を使用します
	詳細は src/nx/sdk/i_plugin.h を参照ください
		`extern "C" NX_PLUGIN_API nx::sdk::IPlugin* createNxPlugin()`
		`{`
		`    // The object will be freed when the Server calls releaseRef().`
		`    return new Plugin();`
		`}`

Engineクラスの構成

- 下記コード例はMetadata SDK内のサンプルコードから引用しています
  - 出展：samples/sample_analytics_plugin/src/nx/vms_server_plugins/analytics/sample/engine.cpp

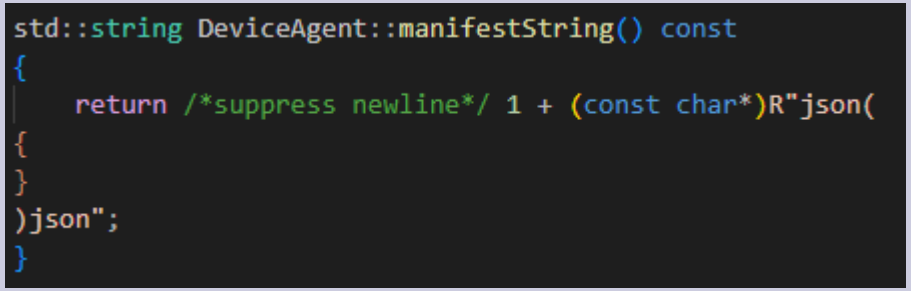
関数名	役割	コード例

Engine::doObtainDeviceAgent ()	プラグインに対応するDeviceAgentクラスのインスタンスを返却します	`void Engine::doObtainDeviceAgent(Result<IDeviceAgent*>* outResult, const IDeviceInfo* deviceInfo)`
		`{`
		`  *outResult = new DeviceAgent(deviceInfo);`
		`}`

Engine::manifestString()	本分析機能のマニフェスト情報をJSON形式で返却します
	記載内容はSDK内の
	src/nx/sdk/analytics/manifests.md
	Engine Manifestの章を参照ください
	
	ユーザーが入力する分析機能の設定スキーマはマニフェスト情報のSettingsModelに指定します
	詳細は
	src/nx/sdk/settings_model.md
	を参照ください
		`std::string Engine::manifestString() const`
		`{`
		`  // Ask the Server to supply uncompressed video frames in YUV420 format (see`
		`  // https://en.wikipedia.org/wiki/YUV).`
		`  //`
		`  // Note that this format is used internally by the Server, therefore requires minimum`
		`  // resources for decoding, thus it is the recommended format.`
		`  return /*suppress newline*/ 1 + (const char*) R"json(`
		`{`
		`  "capabilities": "needUncompressedVideoFrames_yuv420"`
		`}`
		`)json";`
		`}`

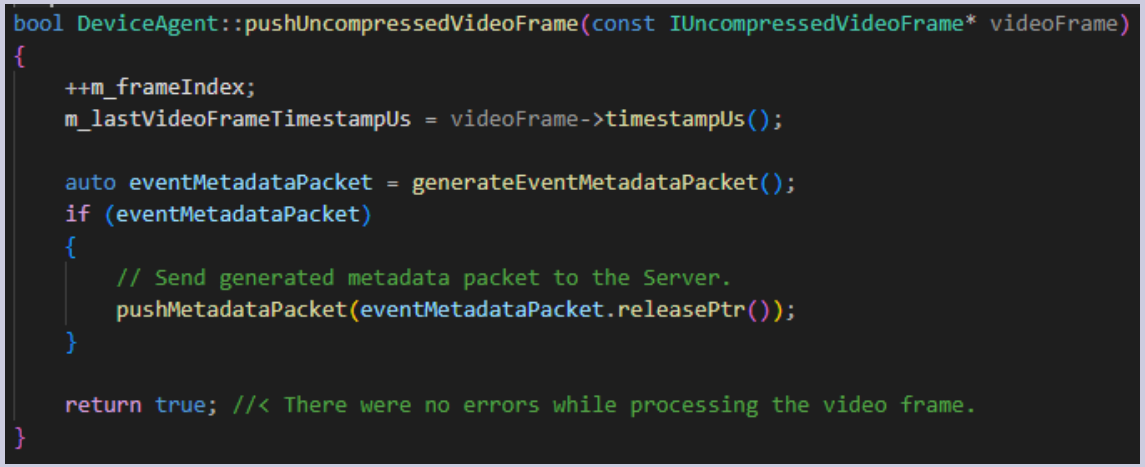
DeviceAgentクラスの構成：DeviceAgentの設定

- 下記コード例はMetadata SDK内のサンプルコードから引用しています
  - 出展：nxsamples/stub_analytics_plugin/src/nx/vms_server_plugins/analytics/stub/settings/device_agent.cpp

関数名	役割	コード例
DeviceAgent::manifestString()	本DeviceAgentの情報をJSON形式で返却します 情報の記法はSDK内の以下を参照ください src/nx/sdk/analytics/manifests.md	
DeviceAgent::settingsReceived()	分析機能の設定(Engineのマニフェスト情報内で設定したSettingsModelに対応する設定)がロード・変更された場合に呼び出されるコールバック関数です ユーザが入力した設定内容は settingValue()で取得可能です	コードが長いため割愛

DeviceAgentクラスの構成：カメラ映像の取得

- 下記コード例はMetadata SDK内のサンプルコードから引用しています
  - 出展：samples/sample_analytics_plugin/src/nx/vms_server_plugins/analytics/sample/device_agent.cpp

関数名	役割	コード例
DeviceAgent:: pushUncompressedVideoFrame()	DeviceAgentに関連付けられたカメラの画像(デコード後)を受け取るコールバック関数です 画像に関する情報は引数のvideoFrameに格納されています また、本関数とは別の関数を実装することによりデコード前の映像ストリームを受信することも可能です 詳細は metadata_sdk/src/nx/sdk/analytics/helpers/consuming_device_agent.h の pushCompressedVideoFrame() を参照ください	

```cpp
bool DeviceAgent::pushUncompressedVideoFrame(const IUncompressedVideoFrame* videoFrame)
{
    ++m_frameIndex;
    m_lastVideoFrameTimestampUs = videoFrame->timestampUs();

    auto eventMetadataPacket = generateEventMetadataPacket();
    if (eventMetadataPacket)
    {
        // Send generated metadata packet to the Server.
        pushMetadataPacket(eventMetadataPacket.releasePtr());
    }

    return true; //< There were no errors while processing the video frame.
}
```

DeviceAgentクラスの構成：メタデータ(オブジェクト・イベント)の登録

- 下記コード例はMetadata SDK内のサンプルコードから引用しています
  - 出展：samples/sample_analytics_plugin/src/nx/vms_server_plugins/analytics/sample/device_agent.cpp

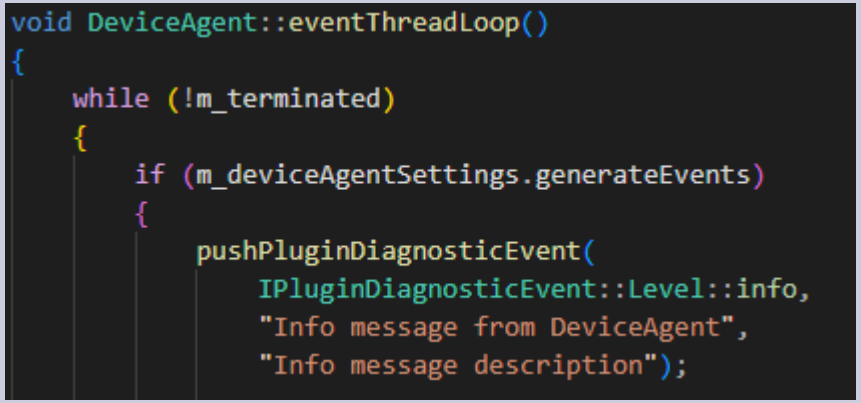
関数名	役割	コード例

DeviceAgent::
pushMetadataPacket()
	引数に指定したメタデータをNx本体に登録します
	
	オブジェクトを登録する場合はObjectMetadataPacketクラス、イベントを登録する際はEventMetadataPacketクラスのインスタンスを引数に設定します
	
	一度に複数のメタデータを登録可能なpullMetadataPackets() も使用可能です。(詳細はmetadata_sdk/src/nx/sdk/analytics/helpers/consuming_device_agent.h を参照ください)
		`bool DeviceAgent::pushUncompressedVideoFrame(const IUncompressedVideoFrame* videoFrame)`
		`{`
		`  ++m_frameIndex;`
		`  m_lastVideoFrameTimestampUs = videoFrame->timestampUs();`
		
		`  auto eventMetadataPacket = generateEventMetadataPacket();`
		`  if (eventMetadataPacket)`
		`  {`
		`  // Send generated metadata packet to the Server.`
		`  pushMetadataPacket(eventMetadataPacket.releasePtr());`
		`  }`
		
		`  return true; //< There were no errors while processing the video frame.`
		`}`
		
		メタデータを生成するコード例は generateEventMetadataPacket() 及び generateObjectMetadataPacket() を参照ください

DeviceAgent::
pullMetadataPackets()
	メタデータを登録するためのコールバック関数です
	
	pushMetadataPacket()とは異なり、Nx本体が本関数をコールし、得られた情報を自動的に登録します
		`bool DeviceAgent::pullMetadataPackets(std::vector<IMetadataPacket*>* metadataPackets)`
		`{`
		`  metadataPackets->push_back(generateObjectMetadataPacket().releasePtr());`
		
		`  return true; //< There were no errors while filling metadataPackets.`
		`}`

DeviceAgentクラスの構成：診断イベントの登録

- 下記コード例はMetadata SDK内のサンプルコードから引用しています
  - 出展：samples/stub_analytics_plugin/src/nx/vms_server_plugins/analytics/stub/diagnostic_events/device_agent.cpp

関数名	役割	コード例
DeviceAgent:: pushPluginDiagnosticEvent()	引数のデータを診断イベントとしてNx本体に登録します 以下のデータを指定します • ログレベル(info/warning/error) • 見出し文字列 • 説明文字列	

```
void DeviceAgent::eventThreadLoop()
{
    while (!m_terminated)
    {
        if (m_deviceAgentSettings.generateEvents)
        {
            pushPluginDiagnosticEvent(
                IPluginDiagnosticEvent::Level::info,
                "Info message from DeviceAgent",
                "Info message description");
```

参考URL: Error Handling in the Plugin

実装のベストプラクティス・注意点

- 可能な場合は検出枠(bounding box)を登録すること
  - オブジェクトの座標を登録することで、スマートサーチ(映像上の特定の部分を指定し、この部分に関連する事象を検索できる機能)を利用可能となります
- 処理遅延を少なくすること
  - Nxクライアントアプリは処理遅延対策のために500msのバッファを持っているため、処理遅延が300ms以内であれば見た目の問題は生じません
    遅延がそれ以上の場合は録画映像上で閲覧はできますが、リアルタイムでの閲覧は動作しません
- 全ての映像を処理できない場合は映像処理をキューイングするのではなく破棄すること
  - 映像を保持するためのメモリ使用量が増大し、Nx本体がクラッシュする可能性があります
  - もし実装する場合はキューサイズを固定とし、300ms以上前のフレームは破棄するのが望ましいです
  - なお、Nx本体はプラグインに対して全ての映像フレームを送信する仕様となっています
- 可能な限りオブジェクトだけでなくイベントを登録すること
  - イベントを登録することでNxに組み込まれているイベントルール機能を使用可能となり、外部装置・システム連携が可能となります
- フレームごとにはイベントを登録しないこと
  - ユーザーにとっても大きな価値はなく、イベントが大量に登録されるとシステムが過負荷になります
- イベントには正確なタイムスタンプを使用すること
  - 正確なイベント発生時刻を特定できるように、特に映像分析処理に時間がかかる場合には処理後の時刻のタイムスタンプではなく、
    映像に付与されたタイムスタンプを使用するのが望ましいです

参考URL: Implementation Best Practices
Important Technical Considerations w/ Nx Witness

その他TIPSなど

- ビデオ(.avi, .mp4ファイル)をカメラの映像として利用する
  - testcamera : 使用法のリンク
- プラグイン開発時のデバッグ方法
  - Plugin Debugging
- プラグインが外部ライブラリに依存する場合の注意点
  - SDK内の src/nx/sdk/dynamic_libraries.md に記載があります

公開ドキュメント

- Integrating Video Analytics and Metadata
  - プラグイン開発全般に関して記載されており、チュートリアルや開発例が記載されています
    (開発環境は弊社ゲートウェイと異なる場合があります)
- Developer Forum
  - Network Optix社が運営するコミュニティサイト内の開発者向けフォーラムです
    ユーザー登録すれば質問を投稿することも可能です
- SDK内のドキュメント
  - docs/html/pages.html にHTML形式のドキュメントがあります