文字化けが。。。

どうすれば解消できるか調べてもらってよいですか？

おそらく、リポジトリがutf-8なのに対して、ソースがSJISであることが原因かと思われます。

> 日本語のままなので、英語に修正しました。

関数の引数についてですが、

変更がされない変数は const参照渡し

変更がされる変数は 参照渡し

としてください。

→ほかの関数も全て見直しをお願いします。

＞　了解です。すべての引数を修正しました。

メンバ変数の命名規則は統一しましょう。

頭にm\_をつけるようにしてください。

→ほかのクラスも対応お願い致します。

＞　了解です。すべての変数を修正しました。

名前をキーとして、様々な情報をmapで持つのは面白い発想です。

しかし、DBとフィーチャクラスとフィールド名はそれぞれ

それぞれ関連付いているものなので、関連付きがわかるような持ち方のほうがベターです。

例えば、POSTALPOINT側のフィールド名とPOI側のフィールド名が万が一同じで有った場合、処理が正常に動かないかと思います。

＞フィーチャクラスを呼ぶとき、関連のDBキーワードを入れることも必要です。フィーチャクラスは混ざることがありません

＞そして、本ツールはただオプションからフィールド名を保存するだけなのでフィールド同じあっても問題ありません。

doEditがfalseになることはあるのでしょうか？

＞あります。POILocationImproverにPOSTALPOINTに対してm\_dataManager->setNewDB(POSTALPOINT\_DB, PPServerName, isPP\_FGDB, !doEdit);をしておいています。POI\_INFOに対してdoEdit必要で、doEditにしています。

編集しないときにRONLYにするのは良いのですが、

ハードコーディングにしてしまうのは、良くありません。

外から指定できるようにしましょう。

そもそも、RONLYでの接続は必要なのでしょうか？

＞以上の理由で!doEditの場合はRONLYします。本ツールのDBは指定されるのでツール以内でRONLY指定しても損はありません。

CStringの変数にserverName等を入れていますが、serverName等をstd::wstringで定義する必要があるのでしょうか？最初からCStringを使用したほうが良いかと思います。

std::stringは文字列がマルチバイトである必要がある場合、std::wstringは文字列がユニコードである必要がある場合のみ使用するようにしてください。

あと、ビルド時の文字コードの設定がマルチバイトの場合は、CStringはcharの文字列になり、文字コードの設定がユニコードの場合は、wcharの文字列となります。

よって、std::wstringをCStringに入れるということは、良くないです。

＞BoostのOptionライブラリはCStringを受け入れられないため、wstringを使うはめになりました。dataManagerの関数はwstringを受けてツールの標準変数CStringに変換する経緯です。しかし、よく考えると最初からwstringをCStringにしたほうがいいので、引数だけを修正します

|  |  |  |  |
| --- | --- | --- | --- |
| [**http://preon/svn/source/trunk/training/p\_sereeyotin/POILocationImprover/OptionManager.cpp**](http://preon.mr.ipc.pioneer.co.jp/reviewboard/r/6495/diff/1/?file=100623)(Diff revision 1) | | | |
|  |  | **68** | \_IOCtrl->print\_no\_timestamp\_run(\_T("POILocationImprover.exe FILEVERSION:1.0.0.0 PRODUCTVERSION:1.0.0.0")); |

バイナリ名やバージョン名を直接書くのはやめましょう。

展開したWinLibを使えば、リソースファイルから自動で取得できます。

やり方は、先日送ったサンプルプログラムを参考にしてください。(CVersionで検索すれば分かるかと思います。)

オプションをenumにしたことで、オプション名を取ってくるのに処理が必要となってしまい、少し冗長なコードとなってしまっています。

＞個人的には、オプションに関係がある変数などはOptionManager.cppにまとめると、冗長に引き換え管理安くなっています。

ArcObjectsのメソッドに渡す引数は、\_bstr\_tでなく

CComBSTRを使用しましょう。

理由としては、こちらから渡しているライブラリではCComBSTRを使っているため、統一したほうが良いからです。

あと，\_bstr\_tよりCComBSTRのほうが実装が軽いそうです。

ほかの場所も\_bstr\_tのところはCComBSTRに統一してもらえますでしょうか？

＞　了解です。すべての変数を修正しました。

以下のように書くことで、コードを減らせます。

if (! PPCoordinates->insert(・・・).second) {

CString errMsg = ・・・;

:

return ErrorMaanger・・・;

}

＞　了解です。コメントのように修正しました。

OBJECTIDはインデックスを調べなくても取得できます。

if (S\_OK != ipPOIFeature->get\_OID(&objectID))

でOBJECTIDは取得できます。

＞　勉強になりました！その通りに修正しました。

IFeatureCursorPtrをメンバ変数にすると、

使う時に既にSearch済みかどうか判断する必要があります。

IFeatureCursorPtrはメンバ変数にせず、必要なタイミングでSearchをするほうが、分かりやすいコードになります。

＞ 了解です。メンバ変数から変数内に移しました。

ここもget\_OIDでobjectidを取得するようにしましょう。

＞　ここも修正しました。

copyPostalPointCoordinatesがこの関数の前のtestのためにgetTargetPOIRecordでも呼ばれていますが、必要あるでしょうか？

→この関数だけで呼んで、失敗した場合はstoreしなければ良いだけではないでしょうか？

　→storeしなくても、座標が変わっている可能性はあるため、ツール実行後の目視確認は必須となります。

＞ 最新コードはgetTargetPOIRecordの役割は編集前に全てのレコードを編集に問題がないかテストすることです。すべてのレコードが問題ないと合格すればupdateToDatabaseという本格の編集に進めます。できる限り中途半端の編集をさせないためです。

そして、storeは想像したよりややこしいです。すでにプレ編集をして回したIFeatureCursorPtrとIFeaturePtrはもう一度プレ編集しないとSiNDYに反映しません。

そのため、getTargetPOIRecord と似ているようにupdateToDatabase関数を作成しました。

return時にgetTargetPOIRecord()を呼び出し、

更にgetTargetPOIRecord()のreturn時に、

updateToDatabase()を呼び出していることで、

コードが少し分かりづらくなっています。

以下のように書いたほうが、ほかの人にとって分かりやすくなります。

if ( is\_success != getRagetPOIRecord(・・・)) {

エラー処理

}

if ( is\_success != updateToDatabase(・・・)) {

　エラー処理

}

return ErrorManager::RCode::R\_SUCCESS

＞ 了解です。以上のように修正しました。

これらの処理は、インスタンスを生成しているだけなので、コンストラクタとかで実施したほうが、コードがすっきりします。

＞ 了解です。移しました。

m\_errorManagerをなぜPOILocationImproverのメンバ変数にしているのでしょうか？

初期化して、m\_IOManagerに渡しているので、m\_IOManagerで定義すれば良いのではないでしょうか？

＞ 見逃してしまいました。m\_IOManagerを削除しました。

|  |  |  |  |
| --- | --- | --- | --- |
| [**http://preon/svn/source/trunk/training/p\_sereeyotin/POILocationImprover/POILocationImprover.cpp**](http://preon.mr.ipc.pioneer.co.jp/reviewboard/r/6495/diff/1/?file=100625)(Diff revision 1) | | | |
|  |  | **450** | m\_OptionManager->setLogger(m\_IOManager); |

複数のクラスにm\_IOManagerを渡していますが、

こういう場合は、singletonというやり方が効果的ですので、今後のために、勉強してみてください。

これらの処理は、

メインの処理の前の準備だと思うので、

init()関数みたいなものを作成し、別で行ったほうがすっきりします。

＞ 了解です。init()とrun()をきっちり役割を別けました。

関数名にpreがあると、事前処理のように見えてしまいます。

名前を変えましょう。

＞ 了解です。名前を変更しました。