ファイバ動作を削除した **AutoCAD** 環境への移行

まず、将来の Windows OS に加える変更を計画し、Windows 版 AutoCAD を Mac OS X に移植するために、AutoCAD デザイン変更を担当する、AutoCAD 開発チームのシニア アーキテクト Bill Adkison 氏に、このホワイトペーパーを公開できることを感謝します。

この記事は、ADN サイトに投稿されたホワイトペーパーの完全な要約版で、同じ場所に、ObjectARX のサンプルプロジェクトとヘッダファイルが記載されています。ホワイトペーパー自体は、より複雑なシナリオを示すコードの断片が含まれており、また、基礎となる AutoCAD の動作をより詳細に示しています。

AutoCAD の開発チームは、現在、ファイバ動作を削除した AutoCAD の開発・移行作業を進めています。このため、ホワイトペーパーは、現在の AutoCAD 2011 とは異なる AutoCAD 開発中のコード ストリームで実装された動作を引き合いに出す場合があります。この情報を公開する目的は、アプリケーションに影響を与える将来の変更内容を、現段階で認識して頂くためです。これらの変更を考慮して、今後のアプリケーションの機能強化を計画することをお勧めします。

はじめに

お手持ちの ObjectARX アプリケーションで acedCommand() や acedCmd() を使用しているか、ObjectARX や.NET アプリケーションで AcApDocManager のメンバ関数を使って、ドキュメントの作成、クローズ、切り替え処理を実装している、または、AutoLISP の *error* ハンドラが (command) 関数を利用している場合には、ここで紹介する移植要件に適合することになります。

AutoCAD は、「ファイバ」と呼ばれるテクノロジへの依存を除去するため、再構成されつつあります。ファイバ機能は、Windows 7、Vista、XP にも搭載されて正常に動作していますが、マイクロソフトは、そのサポートを既に中止しています。AutoCAD アークテクチャの再構成作業は、AutoCAD 2012 では完了していませんが、新しいアーキテクチャへ移行作業を通じて、このドキュメントで説明する以外の問題点を報告してもらえるよう、ファイバ機能を削除したモードへの切り替え機能を実装しています。

OS X にはファイバ構造は存在しませんが、Mac 版 AutoCAD へアプリケーションを移植する場合には、このドキュメントに記載されている acedCommand() の移植作業が必要となります。

ファイバとは？

ファイバは、コルーチンの動作を容易にするために、Windows NT および Windows 95 以降の Windows OS で提供される機能です。ファイバの詳細な説明は、http://ja.wikipedia.org/wiki/ファイバー_(コンピュータ) を参照してください。ファイバは、スレッド内でネイティブコードの呼び出しスタックをスワップすることで、ドキュメント切り替えとコマンド実行を管理するための非常に強力で簡単な方法を提供します。AutoCAD の既存の実装では、ファイバによる制御アーキテクチャが広く使用されています。ファイバは、ネイティブコードの呼び出しスタックの状態を格納する構造体で、他のファイバへの切り替えが可能です。ここでは、マルチ ドキュメント インターフェイス（MDI）環境で、2 つのドキュメントの切り替え例を考えてみます。

- Document1 はアクティブなドキュメントです。ユーザは、マルチ ステップのコマンドを実行中で、コマンドと Document1 に固有のデータがスタックに格納されています。

- ユーザが Dcoument2 にアクティブなドキュメントを切り替えます。その時点で、Document1 のデータを含むスタックが保存され、スタック ポインタは Document2 のデータを含むスタックに移動します。
- しばらくの間ユーザが Document2 で作業した後、Document1 に戻ります。この時、スタックは Document1 のデータを含む状態に戻され（Document2 のスタックと置き換え）、ユーザはコマンド実行中に中断した時点から Document1 で作業を続けることができます（ドキュメント切り替えによる中断など無かったように）。

これがファイバ切り替えの一般的なシナリオです。しかし、実際はより複雑です。ObjectARX MDI API で頻繁に説明される「アプリケーション コンテキスト」は、実際にはドキュメントに依存せずに中立を維持する個別のファイバです。また、各ドキュメントは、メインの AutoCAD コマンド プロセッサの状態と AutoLISP の状態の少なくとも 2 のファイバを持つため、(command) 関数を使用する AutoLISP が、ネイティブ コードのスタックに保持される独自のコンテキストを乱すことなく、AutoCAD のコマンド プロセッサを同期的に駆動させることができます。同様に、ObjectARX で登録したコマンドも、acedCommand() と acedCmd() を使用して 同期的に AutoCAD のコマンド プロセッサ駆動させるため、独自のファイバを使って実行します。これにより、ある ObjectARX 登録コマンドが、別の ObjectARX 登録コマンドを実行することができます。

AutoCAD でファイバが使用されている 3 つの主な領域は、次のとおりです。

- ObjectARX の acedCommand() / acedCmd() 関数
- AutoLISP の (command) 関数
- ObjectARX と .NET の MDI ドキュメントの切り替え

もちろん、スレッドもそれぞれ関連するすべてのプラットフォーム上でネイティブ コードのスタックを維持します。古いオペレーティングシステムでは、スレッドは今日のファイバと同じ目的で使用されていました。ただし、それはコンポーネントを作成したスレッドでのみで動作を厳密に強制する GUI API ライブラリの導入以前でした。このため、最近のオペレーティング システムでは、スレッドをファイバの代わりに使用することができません。

AutoCAD のファイバ動作のすべては、アプリケーションのメインスレッドで発生します。現在の AutoCAD は、AutoCAD のアプリケーション開発環境の領域外で非常に限定された例外を除いて、シングルスレッド アプリケーションです。

なぜファイバは推奨されないのでしょうか？

マイクロソフトは、Windows のファイバ機能のサポートを終了しています。AutoCAD のファイバ機能は現在でも動作しますが、将来の Windows リリースではファイバが使用できなくなることが予測されるため、AutoCAD のアーキテクチャを再構築する必要があります。

ファイバのない動作環境はアプリケーションに何を意味するのでしょうか？

AutoCAD がファイバを使用せずに実行される場合、そのメイン スレッドで 1 つのネイティブ コードスタックを持つことになります。ファイバ切り替えで行われていた制御の遷移に、数多くの変更を加えなければなりません。構造的な制御フローの変更点は、以下のとおりです。

1. スタックの切り替え時に acedCommand()、acedCmd()、(command) 関数を実行する従来の方法は使用できません。新しい関数は、それがスタック上により多くのネイティブ コードをプッシュして（サブルーチン形式）、または、次に実行されるロジックにメッセージをキューイングした後、呼び出しスタックから戻ることによって制御を移すことができます（コルーチン形式）。 FIBERWORLD システム変数が 0 の場合、acedCommand() と acedCmd() は RTERROR を返します。代替の関数は、サブルーチン形式の実行の acedCommandS() と acedCmdS()、コルーチン形式の実行の acedCommandC() と acedCmdC() です。
2. AutoLISP の (command) 関数は、acedCommandC() の使用によって継続してサポートされます。AutoLISP に関連するネイティブな呼び出しスタックの状態をヒープ構造に保存して、引数を処理するために AutoCAD に戻り、ネイティブ呼び出しスタックを再起動して復元します。 AutoLISP アプリケーション本体には、動作に明らかな違いはありません。しかし、どのコードでも発生する可能性があるため、*error* 関数のオーバーライドで (command) を使用することはできません。通常、最善の策は (command) の (commad-s) への置き換えです。(commad-s) は移植パスの一部として導入される新しい関数です。この関数は、コードの開始から終了まで終わりまでの完全なコマンドを表す引数の順序を必要とし、現在はユーザとの対話をサポートしていません。必須ではありませんが、(command-s) が使用することができれば、同等の (command) 呼び出しよりもはるかに高速です。最後に、どうしても *error* 関数 から(command) を使用す必要がある場合には、その方法を選択することもできます。ただし、この方法は複雑になり、推奨しません。
3. 直前にアプリケーション コンテキスト（別名 "セッション ファイバ"）で実行されたロジックは、コールスタックのトップ セクションに位置しているため、常にアクティブであるか、新しいシステム入力のポーリング時に、任意ドキュメントのファイバで実行するため使用されるロジックから、直接サブルーチンコールによって呼び出されます。
4. “アプリケーション コンテキスト”の区別は（"ドキュメント コンテキスト" ではないこと）AcApDocManager::isApplicationContext の戻り値で検証できますが、今後はドキュメント コンテキストのロジックから戻ることによって切り替えられたということになります。
5. AcApDocManager::isApplicationContext() が false の場合、別のドキュメントへ切り替えたり、ドキュメントを削除できたりする関数の呼び出しは、切り替えや削除を同期的に実行しません。その代わりに、システムはアプリケーション コンテキストに制御が戻った後にそのようなアクションを完了します。

アプリケーションをファイバのない動作環境に移植

このセクションでは、主な AutoCAD のアドインの種類（AutoLISP、ObjectARX、.NET API）と RealDWG に必要な特定の変更を網羅しています。

AutoLISP

error ハンドラ内で (command) を使用する場合、AutoLISP コードの唯一の移植要件にあてはまります。ほとんどのケースでは、*error*ハンドラ内の (command) を、単に新しい (command-s) 関数へ呼び出しに置き換える必要があります（コードの残りの部分は現状のまま）。ADN Extranet のホワイトペーパーでは、例外的な状況で必要とされる別の複雑なアプローチを説明しています。

error ハンドラ内で (command) を使用していない場合は、移植作業は必要ありません（ (command) の使用は他の箇所では正常に動作するはず）。しかし、AutoCAD がファイバのない動作環境に移行した際

に、動作変化などの問題に直面しないことを確認するため、ファイバのない環境でアプリケーションをテストすることをお勧めします。

(command-s)

(command-s) 関数は (command) 関数の新しいバリエーションで、コマンド引数のコンテンツに関していくつかの制限があるものの、実行スピードが大幅に速くなっています。

(command-s) の "- s" という接尾辞は、"サブルーチン"（"コルーチン"とは対照的に）の略です。　サブルーチン動作に課される制限は次のとおりです。

1. 1 つの (command-s) 関数式に与えられる一連の引数は、コマンド処理全体を表している必要があります。引数がすべて処理された際には、進行中のコマンドはキャンセルされます。
2. (vl-cmdf) 関数と同じように、与えられた全ての引数は AutoCAD に与えられる前に評価されます。対照的に、(command) はその都度、個々の引数の評価を行い、AutoCAD に結果をフィードバックしながら、AutoLISP を 実 行、再開して次の引数の評価に移ります。
3. "PAUSE" 引数は使用することはできません。 ユーザ対話を実施する評価式を使用することもできますが、AutoCAD がそれらの処理を受け取る前にすべて処理されます。

ObjectARX

ObjectARX は、AutoLISP よりも多くの重要な移植要件があります。MDI API と acedCommand() / acedCmd() の使用の 2 つのエリアの移植に注意が必要です。

MDI API の移植

AutoCAD は、ファイバのない環境で実行している場合でも、引き続きアプリケーション コンテキストとドキュメント コンテキストの概念によって動作しています。アプリケーション コンテキストに到るために、スタックの切り替えの代わりに、AutoCAD は論理的にドキュメントにバインドされている関数から戻る必要があります。

アクティブになるドキュメントが制御を引き継ぐ前に、前回、アクティブだったドキュメントが制御を戻さなければならないように、アクティブなドキュメントの切り替えはアプリケーション コンテキストでのみ行うことができます。これは、 AutoCAD ユーザとアプリケーションが、アクティブなコマンドや AutoLISP 式の実行中に、ドキュメント切り替えを経てコマンドや AutoLISP 式を再開できなくなる、ということを意味します。

ファイバのない環境でのドキュメント切り替えは、以前のドキュメント上で実行されていたコマンドや AutoLISP 式の終了後にのみ行われます。つまり、ドキュメント切り替えを意図した関数への呼び出しを行った後には、アプリケーションはそのドキュメントに復帰する必要があります。そのような状態を経ることで、 AutoCAD はドキュメントの切り替えを完了することができます。ドキュメントの切り替え要求には、アクティブなドキュメントを閉じたり、また、新しいドキュメントを開いたり/作成したりする処理も含まれます。これは、関連するいくつかのリアクタのコールバックのタイミングの違いにつながります。

これら関数のファイバ削除による影響の詳細な説明については、ADN Extranet のホワイトペーパー *API Migration Guide for Fiberless Operations (White Paper).doc* を参照してください。

acedCommand（）の移植

acedCommand() の使用には、2 つの移植方法があります。1 つはより簡単であり、約 90%の確率で適合します。もう一方は面倒ですが、acedCommand() がファイバのない環境で実行している時でも、ファイバ使用時と同じ機能を提供します。

移植方法の違いは、acedCommandS() と acedCommand() の 2 つの新たな機能に基づいています。'S' と 'C' の接頭辞は、それぞれ "サブルーチン" と "コルーチン" を意味しています。どの関数で acedCommand() を置き換えるかの選択は、acedCommand() の使用目的により異なってきます。どのような場面で、どのバージョンを使用する必要があるか説明していきますが、まだ概念に慣れていないようなら、最初にサブルーチンとコルーチンの違いの一般的な説明についてウェキペディアの記事（http://ja.wikipedia.org/wiki/コルーチン）を一読することお勧めします。

acedCommandS()

acedCommandS() のサブルーチンの規約は、"1 つ、または、それ以上の完全なコマンドの実行する" のように要約することができます。より詳細な説明を加えるなら、"最初の入力引数が コマンド プロンプトに提供されてコマンドを開始し、最後の入力引数でコマンドを完了する" となります。このアーキテクチャ変更の利点は、acedCommandS() がイベント ハンドラを含み、acedCommand() / acedCmd() より広いコンテキスト環境下で安全に使用できる、という点です。

1 つの引数が、コマンドの 1 つの区切りとなります。このような呼び出しでは：

```
acedCommand(RTSTR, _T("_Line"), RTSTR, _T("0,0"), RTSTR, _T("111,111"), RTSTR, _T(""), RTNONE);
```

`RTSTR, _T("_Line")` が最初のコマンド引数であり、`RTSTR, _T("")` が最後の引数となり、LINE コマンドを終了します。

既存アプリケーションの acedCommand() 呼び出しが、このような規約に沿って実装されている場合には、acedCommandS() を使用して移植をしてください。上記のコードを acedCommandS() で移植するなら、最後に S を追加するだけです。

```
acedCommandS(RTSTR, _T("Line"), RTSTR, _T("0,0"), RTSTR, _T("111,111"), RTSTR, _T(""), RTNONE);
```

しかし、引数を渡してコマンドを開始しても、それを終了しない場合には acedCommandS() は適しているとは言えません。引数が不足するような場面では、実行中のコマンドはキャンセルされます。部分的に完了したコマンドの状態はロールバックされません。たとえば、次の行は、終点の入力要求をしないので、データベースには新しい線分は追加されません。

```
acedCommandS(RTSTR, _T("Line"), RTSTR, _T("0,0"), RTNONE);
```

acedCommandS() の利用が acedCommandC() よりも容易であることを考慮すると（後述）、可能な限り acedCommandS() を使用することで、時間の節約しながらコードを移植することができます。

移植時に ObjectARX 登録コマンドで acedCommandS() を使用すると、FIBERWORLD システム変数の設定[1] とは関係なく動作します。acedCommandS() と acedCmdS() は、AutoCAD 2012 で実装されています。

acedCommandC()

acedCommand() のコルーチンの規約は、“AutoCAD にいくつかの引数（部分的なコマンド）提供してから、それらを処理できるように AutoCAD に制御を与え、処理終了後に AutoCAD からアプリケーションに制御が返されることで、アプリケーション側で処理を継続することができる。同じ連続コマンド プロセッサ コンテキストで処理されるトークンを順次送信することで、必要に応じてこれを繰り返すこと

ができる" と要約することができます。言い換えれば、acedCommand() 呼び出しでユーザとの対話を一旦停止する処理を実装している場合には、acedCommandC() を使用する必要があります。

ファイバが有効な場合、処理を継続するために AutoCAD に制御を戻す作業は、必要に応じてファイバを切り替える acedCommand() ロジック内で行われます。ファイバを無効にした場合には、ObjectARX コマンドや AutoLISP は、実際にそれらの呼び出し元のドキュメントに戻る必要があります。 AutoCAD は、ObjectARX コマンドや AutoLISP のロジックを続行できるように提供されるコールバック関数を呼び出します。

コルーチンのアプローチを必要とする簡単な acedCommand() の使用例は、ADN サイトにホワイトペーパと共に掲載されている acedCommandTests のサンプルプロジェクトの TYPE2 コマンドで実装されています。

```
static void adskacedCommandTeststype2(void)
{
	acedCommand(RTSTR, _T("Line"), RTSTR, _T("0,0"), RTSTR, _T("111,111"), RTNONE);
	while(isLineActive())
		acedCommand(RTSTR,PAUSE,RTNONE);
	acutPrintf(_T("\nFinished LINE command - control returned to addin\n"));
}
```

（isLineActive はコマンドがまだアクティブであるかどうかを確認する簡単なヘルパー関数です）。

上記の実装を acedCommandC() を使用したコードに移植すると、次のコードのようになります。（簡素化のため、ここではコメントは削除されていますが、acedCommandTests_Migrated プロジェクトにはコメントが含まれています）：

```
static void myCallbackFn1(void * pData)
{
	if (isLineActive())
		acedCommandC(&myCallbackFn1, NULL, RTSTR,PAUSE,RTNONE);
	else
		acutPrintf(_T("\nFinished LINE command - control returned to addin\n"));
}

static void adskacedCommandTeststype2(void)
{
	acedCommandC(&myCallbackFn1, NULL, RTSTR, _T("Line"), RTSTR, _T("0,0"), RTSTR,
_T("111,111"), RTNONE);
}
```

注：テストで使用した AutoCAD 2011 64 ビット版では、移植したプロジェクトの動作が不安定であることが判明しています。従って、AutoCAD 2011 を使ってテストする場合は、32 ビット製品で移植テストすることをお勧めします。AutoCAD 2012 を利用する場合には、問題はありません。

移植した TYPE2 コマンドの動作には違いがあります。ユーザが Esc キーを押して LINE コマンドを終了した場合でも、（古い）acedCommand() のバージョンでは制御がカスタム コマンドの実装関数に戻ります。 （新しい）acedCommandC() のバージョンでは、Esc キーを押すとコマンド自体をキャンセルします。 - LINE コマンドが Enter キーで終了した場合のみ、 acutPrintf() が実行されます。

TYPE3 コマンドの実装は、別の（あまり一般的ではない）コルーチンの使い方を示します。

```
static void adskacedCommandTeststype3(void)
{
	acedCommand(RTSTR, _T("Line"), RTSTR, _T("0,0"), RTSTR, _T("111,111"), RTNONE);
	for (int i=0; i<3; ++i)
		acedCommand(RTSTR, PAUSE, RTNONE);
	acedCommand(RTSTR, _T(""), RTNONE);
```

```
        acutPrintf(_T("\nFinished LINE command - control returned to addin\n"));
}
```

ここでは、1 つの線分を作図して、ユーザに 3 つの線分を追加して作図するようにしています。

この内容を acedCommandC() を使って移植すると、次のようになります。

```
static void myCallbackFn2(void * pData)
{
        struct MyData *pCountData = reinterpret_cast<struct MyData *>(pData);
        if (pCountData->mCount < 3)
        {
                acedCommandC(&myCallbackFn2, reinterpret_cast<void *>(pCountData), RTSTR,
PAUSE, RTNONE);
        }
        else if (pCountData->mCount == 3)
        {
                acedCommandC(&myCallbackFn2, reinterpret_cast<void *>(pCountData), RTSTR,
_T(""), RTNONE);
        }
        pCountData->mCount++;
}

static void adskacedCommandTeststype3(void)
{
        static struct MyData countData;
        countData.mCount = 0;
        acedCommandC(&myCallbackFn2,    reinterpret_cast<void    *>(&countData),    RTSTR,
_T("Line"), RTSTR, _T("0,0"), RTSTR, _T("111,111"), RTNONE);
}
```

AutoLISP で呼び出し可能な ObjectARX 関数を含む、より複雑な使用例は、ホワイトペーパーで説明されています。

.NET API

acedCommand() は、.NET API では公開されていません。これには理由があります。.NET Framework は、ファイバをサポートしたことがありませんが、.NET から P/Invoke を使って acedCommand() を間接的に呼び出している場合には、移植後に問題が発生します。このため、現在、P/Invoke で関数を呼び出している場合のみ、アプリケーション実装の変更が必要になります。

.NET アプリケーションにとって、ファイバの削除には大きな利点があります。これは、acedCommandS() が .NET API で直接公開することが可能になるためです。（注：執筆の時点ではまだ行われていません）

最後に、ObjectARX セクションに記載した AcApDocManager のメンバ関数を使った MDI API の使用方法は、.NET API にも適用されます。

RealDWG

RealDWG ライブラリは、ファイバを認識していません。RealDWG ライブラリに限定した処理であれば、ファイバが無効となる AutoCAD 上での動作をサポートする移植作業は必要ありません。

acedCommand() / acedCmd() でコンパイル エラーを発生するプロジェクト設定

acedCommand() がファイバがない環境で呼び出された場合、RTERROR を返します。必須ではありませんが、aecdCommand() や acedCmd() への参照をコンパイル時にキャッチしたいなら、ACAD_NOFIBER シンボルを定義してください。この値は、acedads.h のプリプロセッサ ディレクティブで使用されています：

```
// #define ACAD_NOFIBER 1  uncomment or otherwise define if you want to flag
//                         acedCommand/acedCmd usage at compile time.

#ifdef ACAD_NOFIBER
#define acedCommand MustSwitchTo_acedCommandC_or_acedCommandS - - !
#define acedCmd MustSwitchTo_acedCmdC_or_acedCmdS - - !
#else
/* The following functions are supported in ObjectARX */
int             acedCommand (int rtype, ...);
int             acedCmd (const struct resbuf *rbp);
#endif
```

プロジェクト内で ACAD_NOFIBER を定義するか、上記ヘッダ ファイル内で #define 行のコメントを解除して下さい。

移植したアプリケーションのテストする

Windows 版 AutoCAD 2011 を使って、ファイバのない擬似環境に切り替える手順は次のとおりです。

- AutoCAD 2011 を起動します。
- FIBERWORLD システム変数の値を確認します。
- ファイバを使用している場合は、FIBERWORLD= 1 となります。
- NEXTFIBERWORLD システム変数の値を 0 に変更します（FIBERWORLD=1 でファイバを使用中の場合）。
- AutoCAD を一旦終了して再起動します。あるいは、AutoCAD のすべてのドキュメントを閉じて、新たにドキュメントを開くか、作成します。
- FIBERWORLD 変数が 0 であることを確認します。

このように、AutoCAD 2011 以降のバージョンでは、ファイバを削除した想定でアプリケーションをテストする準備が整っています。ただし、NEXTFIBERWORLD の値を 1 に変更しても、AutoCAD を再起動するまでは、ファイバが無効な状態が維持される点に注意してください。（常に変更が加えられたことを確認して、再起動後 FIBERWORLD の値を確認してください）。

注：AutoCAD が起動した際に値が 1 の場合は、値の変更後にすべてのドキュメントを閉じることで、ファイバが無効な状態とファイバが有効な状態を切り替えることができます。AutoCAD を起動した際に FIBERWORLD の値が 0 の場合、ファイバを有効化するために、AutoCAD を再起動する必要があります。

ADN サイトのホワイトペーパーと共に掲載されている acedCommandTests ObjectARX サンプルをコンパイルしてから、AutoCAD にロードして FIBERWORLD= 1 環境下で実行すると、サンプルで定義されたカスタムコマンドが正常に動作します。コマンドは次のとおりです。

- TYPE1 -完全なコマンド引数のセットを使用して 1 つの線分を作図します。
- TYPE2 - 線分を作図し、ユーザがコマンドを終了してアプリケーションに戻るまで、LINE コマンドをアクティブな状態にします。
- TYPE3 - 線分を作図し、アプリケーションに制御を戻す前にユーザがさらに 3 つの線分を作図することができます。

同じアプリケーションを FIBERWORLD= 0 環境で実行すると、acedCommand() の呼び出しで何も起こらないことがわかります。これは、acedCommand() の動作が停止してしまうためです。

acedCommandTests_Migrated プロジェクトには、FIBERWORLD= 0 で動作する移植済の関数が含まれています。

想定される問題点

現時点では、acedCommandS() と acedCmdS() 呼び出しで RTPAUSE 引数をサポートすることが目標ですが、これはまだ達成されていません（執筆時の AutoCAD 2011 上で）。AutoLISP の (command-s) 関数で PAUSE 引数を使用することができますが、これは acedCmdS() を呼び出す前に AutoLISP で処理されるためです。

AutoCAD Windows 版でファイバを無効化したアーキテクチャは現在実装作業中であるため、acedCommandS() / acedCommandC() スタイルの変更に加えて、実装が完了していないことで問題が発生することも予想されます。

FIBERWORLD= 0 の動作モードは、部分的にしか AutoCAD 2011 に実装されておらず、多くの不具合修正が実施されつつあります。実際の作業環境では、ファイバが無効なアーキテクチャを使用しないでください。あくまで、テスト時のみに使用してください。

- PARTIALOPEN[部分的に開く] コマンドのファイバのない環境での実行は、現在サポートされていません。最終的にはサポートされる予定です。
- Visual LISP 対話型統合開発環境（IDE）は、まだ完全にはファイバのないモードをサポートしていません。
- acedCommandC() と acedCommandS() の両関数は、様々な理由でハングアップを誘発する可能性があります。修正は、AutoCAD 2011 以降に順次行われる予定です。
- ユーザへのプロンプト表示が AutoLISP の (command) 関数で実行されている場合、ESC キーが押されると、システムがハングアップまたはクラッシュすることがあります。

次へのステップは？

2012 年 10 月現在、ファイバ技術の最終的な除去対象は未定です。この情報を公開する目的は、アプリケーションに影響を与える可能性のある将来的な変更の内容を認識して頂くためです。AutoCAD 2013 以前のバージョンが持つ (command)/ acedCommand() / acedCmd() の幾つかの強力な振る舞いは、ファイバのない環境では使用できなくなることを念頭に置いて、今後のアプリケーションの機能強化を計画することをお勧めいたします。

AutoCAD 2013 以前のアプリケーションを開発、リリースする場合には、ファイバの無効化に対応したアプリケーションを出荷しないように注意して下さい。

もし、ファイバが存在しない OS X にアプリケーションの移植を計画している場合は、これらの変更は移行プロセスの一部として必要となります。