

# Protecting Against Bad Parameter Counts

By David Adams  
Technical Note 07-14

## Overview

---

正しくない引数リストを渡してカスタムメソッドをコールすると、エラーが表示されたり、クラッシュしたり、データの扱いが不正になったりします。幸いなことにこれらの問題はシンプルな防御コードを使用して避けることが可能です。このテクニカルノートでは、不正な数の引数を防がなかった場合の4th Dimensionの動作を確認し、次に不正な引数リストを検知、管理する汎用的なメソッドを提供します。

## Example Method and 4th Dimension Behavior

---

不正な引数リストに起因する問題を議論する前に、存在しない引数を参照したときの4th Dimensionの振る舞いについて見ていきましょう。この議論では、以下のメソッド *DisplaySum* を例とします:

```
C_LONGINT($1;$firstValue_I)
C_LONGINT($2;$secondValue_I)

$firstValue_I:=$1
$secondValue_I:=$2

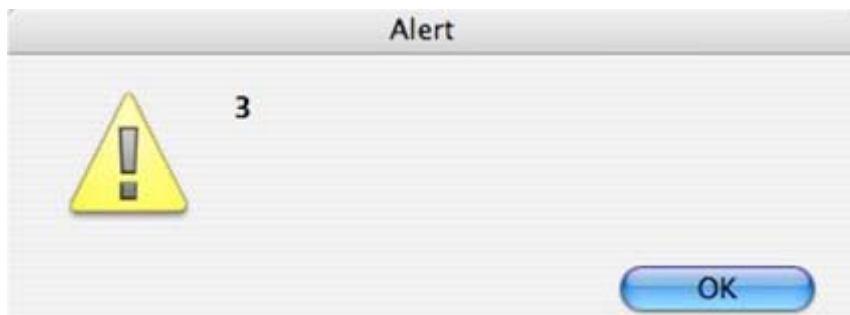
C_LONGINT($sum_I)
$sum_I:=$firstValue_I+$secondValue_I

ALERT(String($sum_I))
```

コードは引数として渡された二つの倍長整数を加算し、結果を表示します。例題コードは意図的に簡単にしています。以下はこのメソッドを正しい引数リストでコールするサンプルです:

*DisplaySum* (1;2)

*DisplaySum*メソッドは\$1と\$2の値を加算し、以下のような結果を表示します:



必要な引数を渡さなかった場合はどうなるでしょうか? 例えば、以下のようにひとつしか引数を渡さないと、4th Dimensionはどうするでしょう:

*DisplaySum* (1) `二つの引数が必要なメソッドに一つだけ渡す

この場合 *DisplaySum* の以下のコードが不正となります:

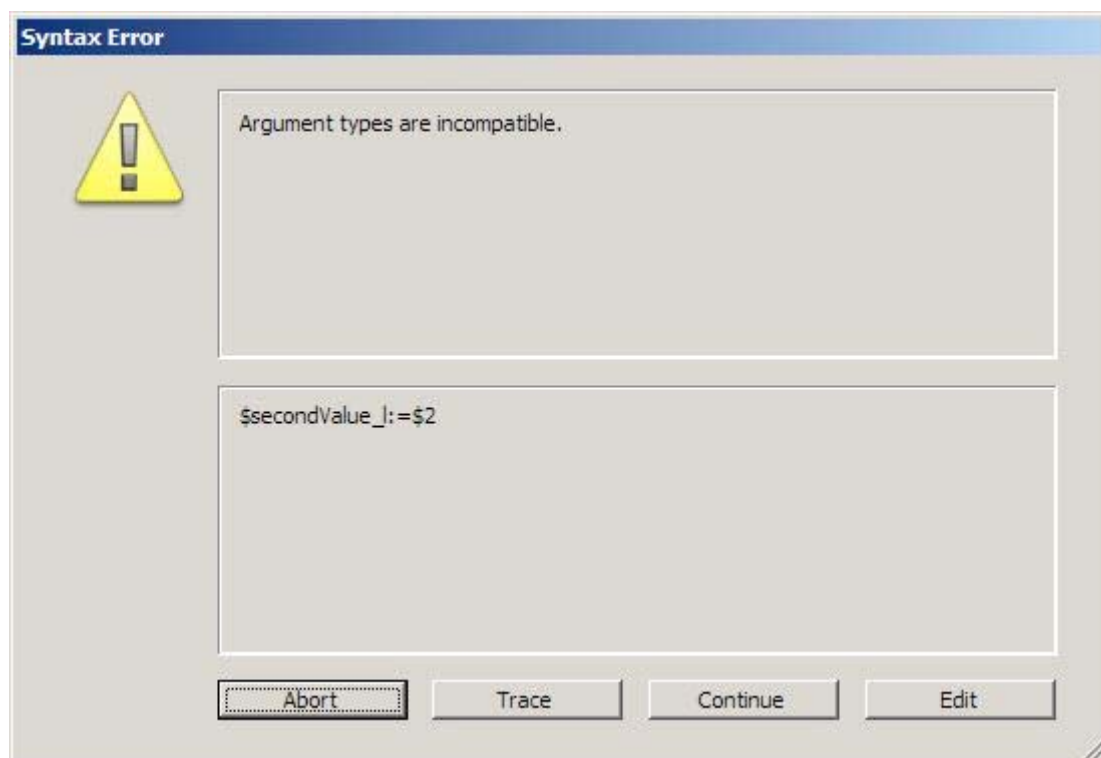
```
$secondValue_1:=$2
```

*DisplaySum* のこの行を実行する際に何が起こるかは、オペレーティングシステムや *4th Dimension* のバージョン、そしてコードをコンパイルしたときのオプションによって異なります。

これより、さまざまなケースで *4th Dimension* がどのような反応を示すか見ていきます。

## Bad Method Calls: Interpreted

インタプリタモードで動作している場合、問題のコードを実行すると以下のようなシンタックスエラーダイアログが表示されます:



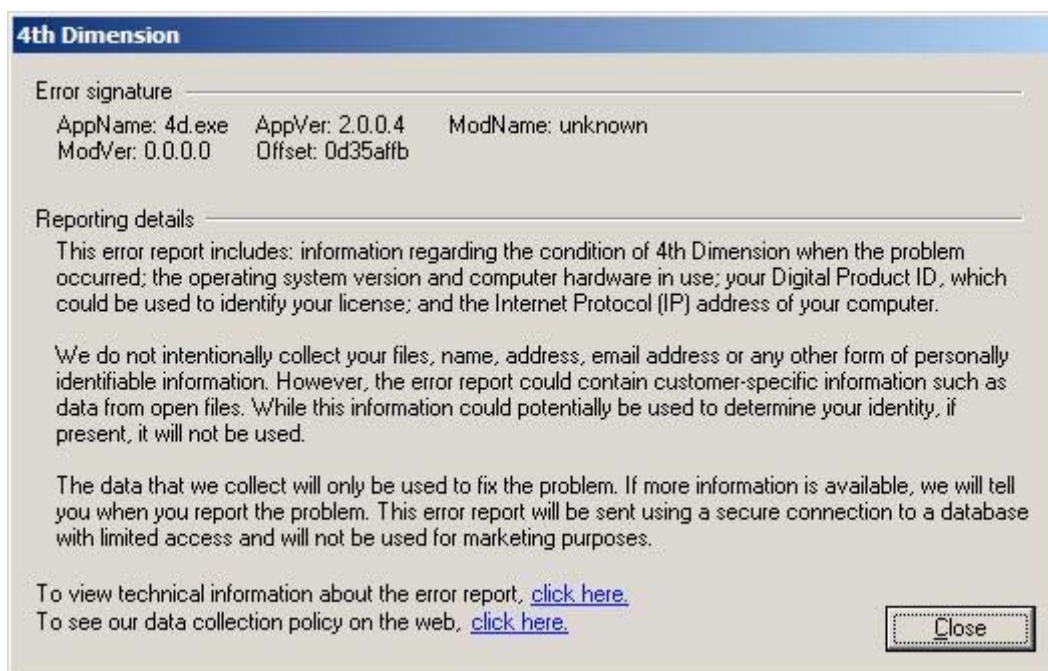
表示されているエラーは、*\$2* に値が渡されなかったことに対する状況を表現しています。首尾一貫して、システムの中の関連するすべてのコードでこの種のエラーを一掃するようにコードを記述することができます。しかし効果のあるテストコードをプロシージャに配置しても、不正な引数リストでメソッドをコールするように書くことは、比較的簡単です。

## Bad Method Calls: Compiled without Range Checking on Windows

不正な引数リストが引き起こす結果は、インタプリタよりもコンパイルされたデータベースでより深刻です。コンパイルされたプログラムでの実際の動作はさまざまですが、いずれも望ましいものとはいえません。例えば以下は、範囲チェックを行わない設定でコンパイルしたデータベースを、*Windows XP* 上で起動し、問題のコードを実行したときに発生するものです:



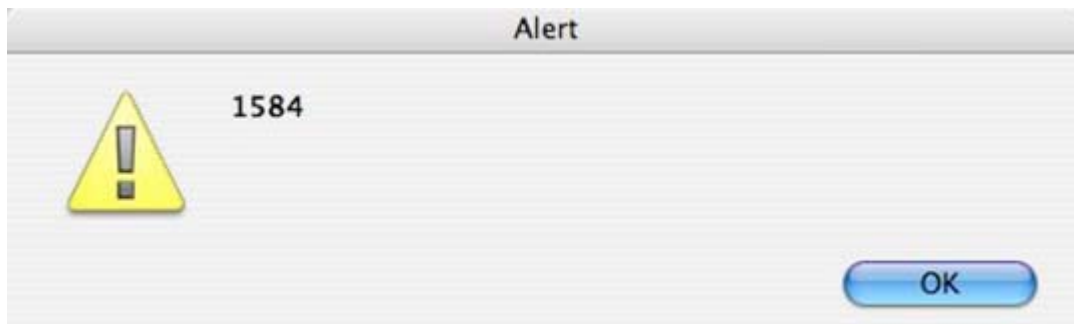
このようなダイアログは誰も見たくないものです。何百、何千行のコードの中で、たった一つの不正な引数リストのために、システムがクラッシュしてしまいます。よってこのような問題に対処することは価値のあることです。特に上記のクラッシュダイアログのように、問題の原因についてほとんど何も教えてくれないような問題が発生するケースではなおさらです。4th Dimensionにバグがあるのか、プラグインか、あるいは開発者のコードに問題があるのか、このダイアログからは知ることができません。例えば、以下の画像は[click here](#) リンクをクリックしたときに表示される技術情報です：



残念なことに、上記エラースクリーンには関連する詳細情報や、問題点を特定するような情報はなにもありません。

## Bad Method Calls: Compiled without Range Checking on OS X

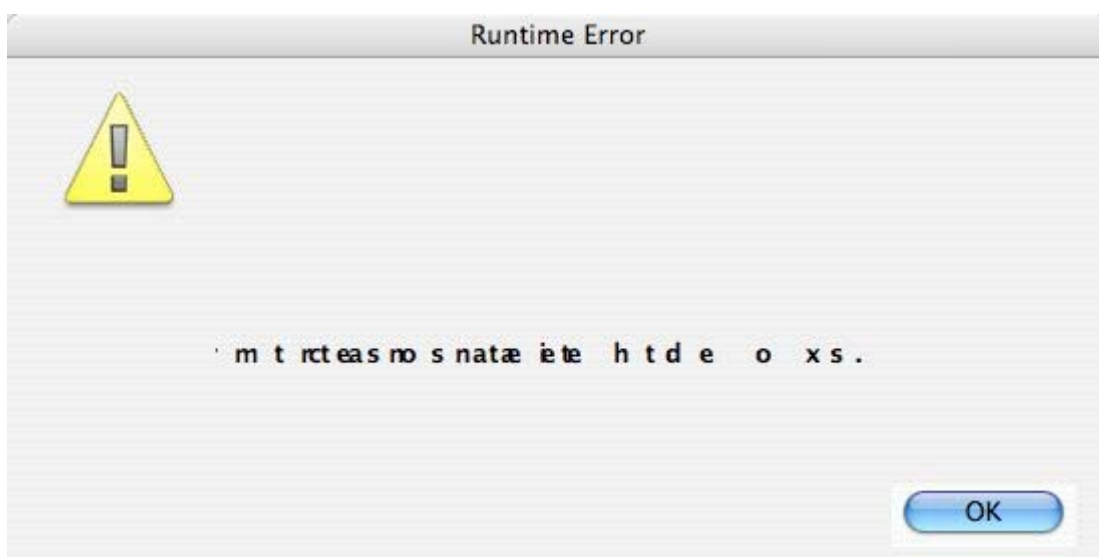
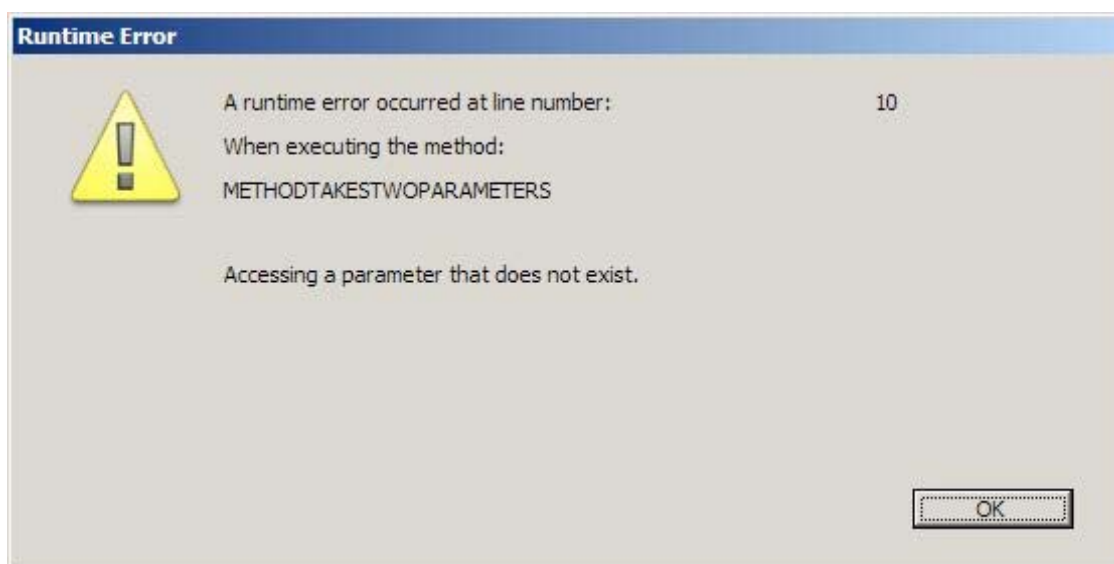
クラッシュが発生すれば、少なくともシステムに問題が発生していることがわかります。範囲チェックなしでコンパイルした問題のコードをMac OS Xで実行すると、問題はよりわかりづらくなります：



今回のシンプルな例では、1が1584と表示され、誤りがあったことがすぐにわかります。しかしコードが複雑な計算を行うものであればどうでしょう？ データは微妙に間違っただけになる可能性があります。これと比べれば、クラッシュが起きたほうが、注意すべき出来事が発生したのが明瞭であり、ましだといえます。

### Bad Method Calls: Compiled with Range Checking

範囲チェックが有効になっていれば、4th Dimensionはインタプリタモードと同じようなダイアログを表示しようとします:



この警告は、クラッシュしたり、誤った結果を表示するよりはより助けとなるものです。OS Xでのダイアログがたとえ読めないものであったとしても、プログラムのどこかがおかしいことがわかります。このようなアラートとは、データが壊されてしまう前に、バグを発見する機械を提供します。

**Tip** コンパイルは範囲チェックをオンにして行う。実行速度の差は小さなものであり、非常にまれなケース以外でそのことを気にする必要はありません。

## About the Behavior Documented Above

今まで説明した、不正な引数リストに対する4th Dimensionの振る舞いをあてにすることはできません。将来のバージョンでは動作が変わることもありえます。

今まで見てきた振る舞いの中には、ほとんど妥当と思われるものはありませんでした。よくて、4th Dimensionは行儀よく実行を失敗することくらいしかできません。

それよりも、不正な引数リストは開発者のバグであり、発見され、修正されるべきものです。

## Detecting and Stopping Bad Parameter Lists

---

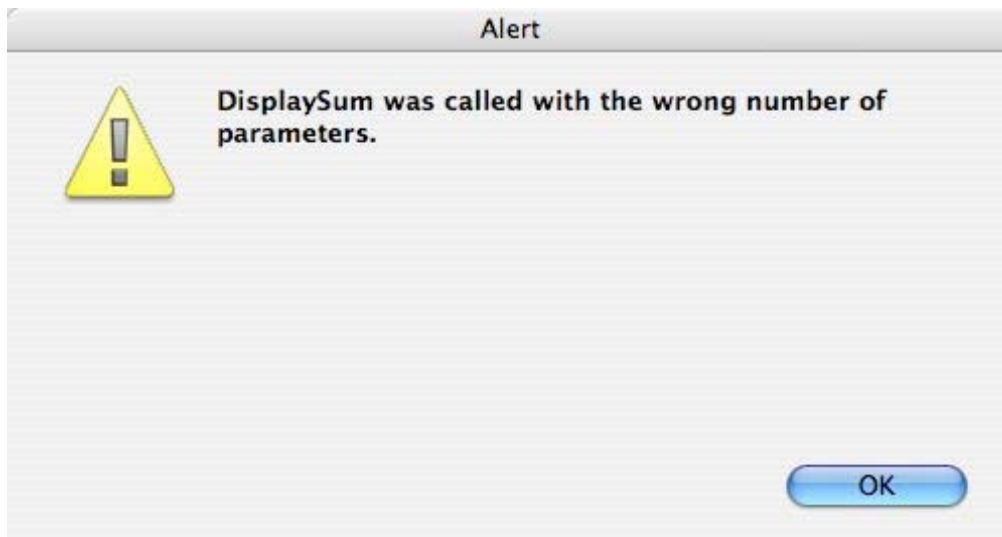
### Counting Parameters within Each Method

失われた引数の参照や代入を防ぐ方法は、それらを使用する前に引数の数を数えることで簡単に確認できます:

```
If (Count parameters=2)
    $firstValue_1:=$1
    $secondValue_1:=$2

    C_LONGINT($sum_1)
    $sum_1:=$firstValue_1+$secondValue_1
    ALERT(String($sum_1))
Else
    ALERT("DisplaySum was called with the wrong number of parameters.")
End if
```

引数を二つ以外渡してこのメソッドをコールすると、このメソッドは以下のようなアラートを表示します:



簡単な**If** テストと**ALERT** がこのコードをよりよいものにしています。このコードではクラッシュや不正な結果が生じることはありません。画面には問題の箇所が明示的に表示されます。

**Tip ALERT** コマンドを使用しないことも可能です。このコマンドを使用できない状況ではエラーをログに記録したり、返り値として返したり、現在のコンテキストにあった方法で処理します。このような状況は例えばトリガや**Web**、**Web**サービスでコードが実行されているときに当てはまります。

## Using a Centralized Parameter Tester

先に示したコードは失われた引数を検知するのに役立ちますが、エラーを処理するためのコードが直接メソッドに記述されています。

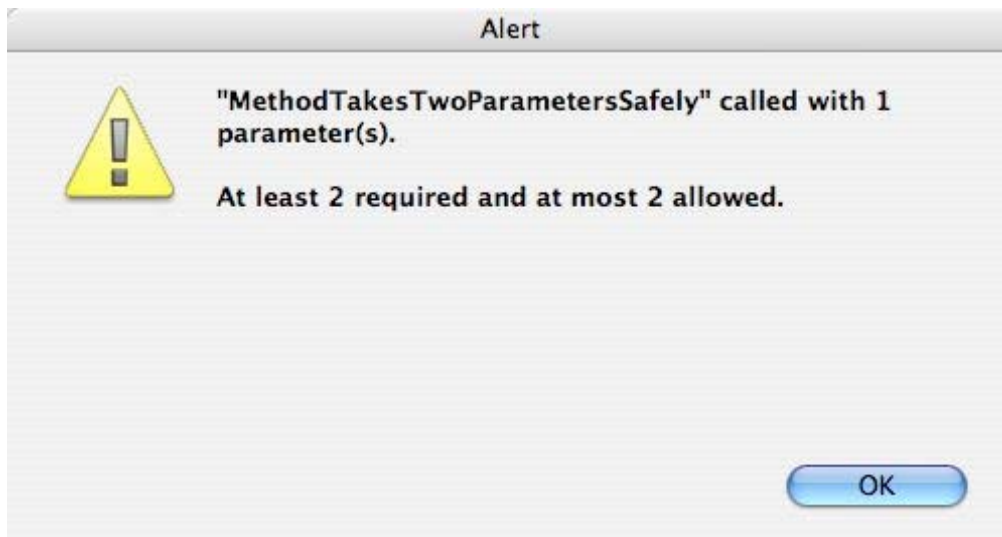
多くの開発者にとって、作業を行うコードとエラーを管理するコードを分離させたほうが便利で自由度が高くなります。例えば以下のメソッドを見てください：

```
C_LONGINT($1;$firstValue_I)
C_LONGINT($2;$secondValue_I)

If (ParameterCountIsOkay (Current method name;2;2;Count parameters))
    $firstValue_I:=$1
    $secondValue_I:=$2

    C_LONGINT($sum_I)
    $sum_I:=$firstValue_I+$secondValue_I
    ALERT(String($sum_I))
End if
```

直接引数をカウントするのではなく、関数**ParameterCountIsOkay**が呼ばれています。サブルーチンは呼び出しもとのメソッド名、必要とされる引数の最小数、必要とされる引数の最大数、そして実際に受け取った引数の数を渡されます。**ParameterCountIsOkay**はこれらの引数をテストし、引数の数が範囲に収まっているか検査します。もし範囲外にあれば、以下のようなダイアログを表示します：



*ParameterCountsOkay* ルーチンのコードは以下のようになります:

```
C_BOOLEAN($0;$countIsOkay_b)
C_TEXT($1;$methodName_text)
C_LONGINT($2;$min)
C_LONGINT($3;$max)
C_LONGINT($4;$count)

$countIsOkay_b:=False` Default.
If (Count parameters>=4) ` Test that this routine has enough parameters!
    $methodName_text:=$1
    $min:=$2
    $max:=$3
    $count:=$4

    If (($count<$min) | ($count>$max)) ` Not enough parameters | Too many parameters
        $countIsOkay_b:=False` This is a good place to log or display an error.

        C_TEXT($error_text)
        $error_text:=""
        $error_text:=$error_text+Char(Double quote)+$methodName_text+Char(Double quote)
        $error_text:=$error_text+" called with "+String($count)+" parameter(s). "
        $error_text:=$error_text+Char(Carriage return)+Char(Carriage return)
        $error_text:=$error_text+"At least "+String($min)+
        $error_text:=$error_text+" required and at most "+String($max)+" allowed."
        ALERT($error_text)

    Else
        $countIsOkay_b:=True
    End if

Else ` The ParameterCountsOkay routine didn't get enough parameters.
    ` This is a good place to log or display an error.
    C_TEXT($error_text)
    $error_text:=""
    $error_text:=$error_text+" called with "+String($count)+" parameter(s). "
    $error_text:=$error_text+Char(Carriage return )
    $error_text:=$error_text+"Four parameters are required"
    ALERT($error_text)
End if `(Count parameters>=4)

$0:=$countIsOkay_b
```

このように関数にロジックを移動することにはいくつかの利点があります:

- 個々のメソッドに対して引数を使用する前に**If**テストを追加することが簡単になります。それぞれのメソッドが個別にエラーを処理する必要がなくなり、その分シンプルになります。



- ・コードのテスト、デバッグ、管理がより簡単になります。

- ・コードを集中させることで、新しい機能の追加が簡単にできます。例えばログ機能の実装やエラーのメール機能、あるいはトリガや **Web/SOAP** プロセスでエラーダイアログを出さないようにするなど、をひとつのメソッドの中で処理できます。

## Regarding ON ERR CALL

**ON ERR CALL** コマンドを使用して、エラー管理処理に独自のメソッドを挿入することができますようになります。エラーが検知されると、**ON ERR CALL**でインストールされたメソッドが実行されます。プログラムはクラッシュせず、範囲チェックダイアログやシンタックスエラーダイアログも表示されません。残念なことに、**ON ERR CALL**にはドキュメント化されていない制限があり、すべてのエラーをトラップするわけではありません。

例えば、**ON ERR CALL** は不正な引数リストによるコンパイルモードでのエラー処理についてはうまく管理できません。

## Summary

---

カスタムメソッドで渡されない引数を参照するとエラーダイアログが表示されたり、クラッシュしたり、結果が誤るなどの現象が発生します。これらの問題は単純のコードで防ぐことが可能です。このテクニカルノートでは、不正な引数リストで引き起こされる問題を確認し、それを完全に避ける方法について紹介しました。