

# Avertissement du compilateur (niveau 3) C4996

• 01/12/2021 • 10 minutes de lecture

Cette page est-elle utile ?  

## Cet article

s

[Activer l'avertissement](#)

[s des fonctions POSIX](#)

[tions de la bibliothèque CRT non sécurisée](#)

[tions de la bibliothèque standard non sécurisée](#)

[eurs vérifiés activés](#)

[MFC ou ATL non sécurisé](#)

[bles et fonctions CRT obsolètes](#)

[haling des erreurs dans le code CLR](#)

[iple : fonction déconseillée définie par l'utilisateur](#)

code utilise une fonction, un membre de classe, une variable ou un typedef marqué *ne déconseillé*. Les symboles sont déconseillés à l'aide d'un `__declspec(deprecated)`ificateur ou de l'attribut `c++ 14 [[deprecated]]`. Le message d'avertissement C4996 est spécifié par le `deprecated` modificateur ou l'attribut de la déclaration.

## Important

Cet avertissement est toujours un message délibéré de l'auteur du fichier d'en-tête qui déclare le symbole. N'utilisez pas le symbole déconseillé sans comprendre les conséquences.

## tes

ombrées fonctions, fonctions membres, fonctions de modèle et variables globales *déconseillées* dans les bibliothèques de Visual Studio. Certains, tels que POSIX et les ions spécifiques à Microsoft, sont déconseillés, car ils ont maintenant un nom préféré ent. Certaines fonctions de la bibliothèque Runtime C sont dépréciées, car elles ne pas sécurisées et ont une variante plus sécurisée. D'autres sont dépréciées, car elles

obsolètes. Les messages de désapprobation incluent généralement un remplacement éré pour la fonction ou la variable globale déconseillée.

## sactiver l'avertissement

résoudre un problème C4996, nous vous recommandons généralement de modifier code. Utilisez à la place les variables globales et les fonctions suggérées. Si vous z utiliser les fonctions ou variables existantes pour des raisons de portabilité, vous ez désactiver l'avertissement.

### sactiver l'avertissement pour une ligne de code cifique

désactiver l'avertissement pour une ligne de code spécifique, utilisez le [warning](#) na `#pragma warning(suppress : 4996)` .

### sactiver l'avertissement dans un fichier

désactiver l'avertissement dans un fichier pour tout ce qui suit, utilisez le pragma ng, `#pragma warning(disable : 4996)` .

### sactiver l'avertissement dans les générations à partir de gne de commande

désactiver globalement l'avertissement dans les générations à partir de la ligne de nande, utilisez l' [/wd4996](#) option de ligne de commande.

### sactiver l'avertissement pour un projet dans Visual dio

désactiver l'avertissement pour l'intégralité d'un projet dans l'IDE Visual Studio :

Ouvrez la boîte de dialogue **pages de propriétés** de votre projet. Pour plus d'informations sur l'utilisation de la boîte de dialogue pages de propriétés, consultez [pages de propriétés](#).

Sélectionnez la page de propriétés avancé des **Propriétés de configurationC/C++** .

Modifiez la propriété **Désactiver les avertissements spécifiques** à ajouter . Choisissez **OK** pour appliquer vos modifications.

## Désactiver l'avertissement à l'aide de macros de préprocesseur

Vous pouvez également utiliser des macros de préprocesseur pour désactiver certains avertissements spécifiques d'avertissements de désapprobation utilisés dans les bibliothèques. Ces macros sont décrites ci-dessous.

Comment définir une macro de préprocesseur dans Visual Studio :

Ouvrez la boîte de dialogue **pages de propriétés** de votre projet. Pour plus d'informations sur l'utilisation de la boîte de dialogue pages de propriétés, consultez [pages de propriétés](#).

Développez **Propriétés de configuration > préprocesseur C/C++**.

Dans la propriété **définitions de préprocesseur** , ajoutez le nom de la macro. Choisissez **OK** pour enregistrer, puis régénérez votre projet.

Pour définir une macro uniquement dans des fichiers sources spécifiques, ajoutez une ligne de préprocesseur `#define EXAMPLE_MACRO_NAME` avant toute ligne qui comprend un fichier d'en-tête.

Voici quelques-unes des sources courantes d'avertissements et d'erreurs C4996 :

## Noms des fonctions POSIX

**POSIX name for this item is deprecated. Instead, use the ISO C and C++ conformant new-name. See online help for details.**

Microsoft a renommé des fonctions POSIX et de bibliothèque spécifiques à Microsoft dans Visual Studio pour se conformer aux contraintes C99 et C++ 03 sur les noms réservés et globaux définis par l'implémentation. *Seuls les noms sont déconseillés, pas les fonctions elles-mêmes.*

Dans la plupart des cas, un trait de soulignement de début a été ajouté au nom de la fonction pour créer un nom conforme. Le compilateur émet un avertissement de désapprobation pour le nom de la fonction d'origine et suggère le nom préféré.

Pour résoudre ce problème, nous vous recommandons généralement de modifier votre

pour utiliser les noms de fonctions suggérés à la place. Toutefois, les noms mis à jour spécifiques à Microsoft. Si vous devez utiliser les noms de fonctions existants pour des raisons de portabilité, vous pouvez désactiver ces avertissements. Les fonctions sont toujours disponibles dans la bibliothèque sous leurs noms d'origine.

Pour désactiver les avertissements de désapprobation pour ces fonctions, définissez la macro de préprocesseur `_CRT_NONSTDC_NO_WARNINGS`. Vous pouvez définir cette macro sur la ligne de commande en incluant l'option `/D_CRT_NONSTDC_NO_WARNINGS`.

## Fonctions de la bibliothèque CRT non sécurisée

**A function or variable may be unsafe. Consider using *safe-version* instead. To disable this deprecation, use `_CRT_SECURE_NO_WARNINGS`. See online help for details.**

Microsoft a déconseillé certaines fonctions et fonctionnalités globales de la bibliothèque standard C++ standard, car des versions plus sécurisées sont disponibles. La plupart des fonctions déconseillées autorisent un accès en lecture ou en écriture non contrôlé aux buffers tampons. Leur utilisation abusive peut entraîner de sérieux problèmes de sécurité. Le compilateur émet un avertissement indiquant que ces fonctions sont déconseillées et suggère la fonction préférée.

Pour résoudre ce problème, nous vous recommandons d'utiliser à la place la fonction ou la fonctionnalité *safe-version*. Parfois, vous ne pouvez pas, à des fins de portabilité ou de compatibilité descendante. Vérifiez avec soin qu'il n'est pas possible de remplacer ou de sécuriser une mémoire tampon dans votre code. Ensuite, vous pouvez désactiver l'avertissement.

Pour désactiver les avertissements de désapprobation pour ces fonctions dans la bibliothèque CRT, définissez `_CRT_SECURE_NO_WARNINGS`.

Pour désactiver les avertissements concernant les variables globales déconseillées, définissez `_CRT_SECURE_NO_WARNINGS_GLOBALS`.

Pour plus d'informations sur ces fonctions et globales déconseillées, consultez [Fonctionnalités de sécurité dans les bibliothèques CRT](#) et [Coffre : bibliothèque Standard](#)


## Fonctions de la bibliothèque standard non

# Surveillée

`std::function_name` : `std::_Unchecked_iterators::_Deprecate` 'Call to `std::function_name` parameters that may be unsafe - this call relies on the caller to check that the values are correct. To disable this warning, use `-D_SCL_SECURE_NO_WARNINGS`. See documentation on how to use Visual C++ 'Checked Iterators'

Visual Studio 2015, cet avertissement apparaît dans les versions debug, car certaines fonctions de modèle de la bibliothèque C++ Standard ne vérifient pas l'exactitude des limites. C'est souvent parce qu'il n'y a pas assez d'informations disponibles pour la fonction pour vérifier les limites du conteneur. Ou, car les itérateurs peuvent être utilisés de manière incorrecte avec la fonction. Cet avertissement vous aide à identifier ces fonctions, car elles peuvent constituer une source de failles de sécurité importantes dans votre programme. Pour plus d'informations, consultez [itérateurs vérifiés](#).

Par exemple, cet avertissement s'affiche en mode débogage si vous transmettez un pointeur d'élément à `std::copy`, au lieu d'un tableau ordinaire. Pour résoudre ce problème, utilisez un tableau déclaré de manière appropriée, afin que la bibliothèque puisse vérifier les étendues du tableau et effectuer la vérification des limites.


 Copier

```

C4996_copyarray.cpp
compile with: cl /c /W4 /D_DEBUG C4996_copyarray.cpp
#include <algorithm>

void example(char const * const src) {
    char dest[1234];
    char * pdest3 = dest + 3;
    std::copy(src, src + 42, pdest3); // C4996
    std::copy(src, src + 42, dest);   // OK, copy can tell that dest is 1234
}
  
```

Plusieurs algorithmes de bibliothèque standard ont été mis à jour pour avoir des versions à double page » en C++ 14. Si vous utilisez les versions à deux pages, la deuxième page nécessite la vérification des limites nécessaires :

 Copier

```

C4996_containers.cpp
  
```

```

compile with: cl /c /W4 /D_DEBUG C4996_containers.cpp
include <algorithm>

// example(
char const * const left,
const size_t leftSize,
char const * const right,
const size_t rightSize)

bool result = false;
result = std::equal(left, left + leftSize, right); // C4996
// To fix, try this form instead:
// result = std::equal(left, left + leftSize, right, right + rightSize); //

return result;

```

exemple illustre plusieurs autres façons dont la bibliothèque standard peut être utilisée vérifier l'utilisation de l'itérateur, et quand une utilisation non vérifiée peut être éreuse :

Copier

```

C4996_standard.cpp
compile with: cl /EHsc /W4 /MDd C4996_standard.cpp
include <algorithm>
include <array>
include <iostream>
include <iterator>
include <numeric>
include <string>
include <vector>

using namespace std;

template <typename C> void print(const string& s, const C& c) {
    cout << s;

    for (const auto& e : c) {
        cout << e << " ";
    }

    cout << endl;
}

int main()

    vector<int> v(16);
    iota(v.begin(), v.end(), 0);

```

```

print("v: ", v);

// OK: vector::iterator is checked in debug mode
// (i.e. an overrun triggers a debug assertion)
vector<int> v2(16);
transform(v.begin(), v.end(), v2.begin(), [](int n) { return n * 2; });
print("v2: ", v2);

// OK: back_insert_iterator is marked as checked in debug mode
// (i.e. an overrun is impossible)
vector<int> v3;
transform(v.begin(), v.end(), back_inserter(v3), [](int n) { return n * 3; });

print("v3: ", v3);

// OK: array::iterator is checked in debug mode
// (i.e. an overrun triggers a debug assertion)
array<int, 16> a4;
transform(v.begin(), v.end(), a4.begin(), [](int n) { return n * 4; });
print("a4: ", a4);

// OK: Raw arrays are checked in debug mode
// (i.e. an overrun triggers a debug assertion)
// NOTE: This applies only when raw arrays are
// given to C++ Standard Library algorithms!
int a5[16];
transform(v.begin(), v.end(), a5, [](int n) { return n * 5; });
print("a5: ", a5);

// WARNING C4996: Pointers cannot be checked in debug mode
// (i.e. an overrun triggers undefined behavior)
int a6[16];
int * p6 = a6;
transform(v.begin(), v.end(), p6, [](int n) { return n * 6; });
print("a6: ", a6);

// OK: stdext::checked_array_iterator is checked in debug mode
// (i.e. an overrun triggers a debug assertion)
int a7[16];
int * p7 = a7;
transform(v.begin(), v.end(),
    stdext::make_checked_array_iterator(p7, 16),
    [](int n) { return n * 7; });
print("a7: ", a7);

// WARNING SILENCED: stdext::unchecked_array_iterator
// is marked as checked in debug mode, but it performs no checking,
// so an overrun triggers undefined behavior
int a8[16];

int * p8 = a8;
transform( v.begin(), v.end(),

```

```

std::make_unchecked_array_iterator(p8),
[] (int n) { return n * 8; });
print("a8: ", a8);

```

Si vous avez vérifié que votre code ne peut pas avoir une erreur de dépassement de tampon, vous pouvez désactiver cet avertissement. Pour désactiver les avertissements pour ces fonctions, définissez `_SCL_SECURE_NO_WARNINGS`.

## Itérateurs vérifiés activés

Le message d'erreur C4996 peut également se produire si vous n'utilisez pas d'itérateur vérifié lorsque `_ITERATOR_DEBUG_LEVEL` est défini sur 1 ou 2. Elle est définie sur 2 par défaut pour les builds de débogage et sur 0 pour les versions commerciales. Pour plus d'informations, consultez [itérateurs vérifiés](#).

F

 Copier

```

C4996_checked.cpp
compile with: /EHsc /W4 /MDd C4996_checked.cpp
#define _ITERATOR_DEBUG_LEVEL 2

#include <algorithm>
#include <iterator>

using namespace std;
using namespace stdext;

int main() {
    int a[] = { 1, 2, 3 };
    int b[] = { 10, 11, 12 };
    copy(a, a + 3, b + 1);    // C4996
    // try the following line instead:
    // copy(a, a + 3, checked_array_iterator<int *>(b, 3));    // OK

```

## Utilisation de MFC ou ATL non sécurisé

Le message d'erreur C4996 peut se produire si vous utilisez des fonctions MFC ou ATL qui ont été dépréciées pour des raisons de sécurité.

Pour résoudre ce problème, nous vous recommandons vivement de modifier votre code



utiliser des fonctions mises à jour à la place.

plus d'informations sur la façon de supprimer ces avertissements, consultez [\\_SECURE\\_NO\\_WARNINGS](#).

## Variables et fonctions CRT obsolètes

`function or variable has been superseded by newer library or operating system functionality. Consider using new_item instead. See online help for details.`

Les fonctions de la bibliothèque et certaines variables globales sont déconseillées, et certaines sont obsolètes. Ces fonctions et variables sont susceptibles d'être supprimées dans une version future de la bibliothèque. Le compilateur émet un avertissement indiquant que ces éléments sont déconseillés et suggère l'alternative préférée.


Pour résoudre ce problème, nous vous recommandons de modifier votre code pour utiliser la fonction ou la variable suggérée.

Pour désactiver les avertissements de désapprobation pour ces éléments, définissez `_OBSOLETE_NO_WARNINGS`. Pour plus d'informations, consultez la documentation pour la fonction ou la variable déconseillée.

## Marshaling des erreurs dans le code CLR

C4996 peut également se produire lorsque vous utilisez la bibliothèque de marshaling. Dans ce cas, C4996 est une erreur, et non un avertissement. L'erreur se produit lorsque vous utilisez `marshal_as` pour effectuer une conversion entre deux types de données qui requièrent une `marshal_as`. Vous pouvez également recevoir cette erreur parce que la bibliothèque de marshaling ne prend pas en charge une conversion. Pour plus d'informations sur la bibliothèque de marshaling, consultez [vue d'ensemble du marshaling](#) et [-+](#).

Cet exemple génère l'C4996, car la bibliothèque de marshaling requiert un contexte pour effectuer la conversion d'un `System::String` en `const char *`.

C4996_Marshal.cpp compile with: /clr C4996 expected <code>include &lt;stdlib.h&gt;</code>	 Copier
--	--

```

.....
include <string.h>
include <msclr\marshal.h>

using namespace System;
using namespace msclr::interop;

int main() {
    String^ message = gcnew String("Test String to Marshal");
    const char* result;
    result = marshal_as<const char*>( message );
    return 0;
}

```

## Exemple : fonction déconseillée définie par l'utilisateur

Vous pouvez utiliser l'attribut `deprecated` dans votre propre code pour avertir les utilisateurs lorsque vous n'êtes plus recommandé d'utiliser certaines fonctions. Dans cet exemple, l'C4996 est généré à deux emplacements : un pour la ligne sur laquelle la fonction déconseillée est déclarée, et une pour la ligne où la fonction est utilisée.

F

 Copier

```

C4996.cpp
compile with: /W3
C4996 warning expected
include <stdio.h>

#pragma warning(disable : 4996)
void func1(void) {
    printf_s("\nIn func1");

}

[[deprecated]]
void func1(int) {
    printf_s("\nIn func2");

}

int main() {
    func1();
    func1(1);    // C4996
}

```

## Contenu recommandé

## Erreur des outils Éditeur de liens LNK2019

propos de l'éditeur de liens Microsoft Visual Studio erreur LNK2019 et comment le diagnostiquer et le corriger en code C et C++.

## Erreur du compilateur C3861

pour savoir plus sur : erreur du compilateur C3861

## Erreur des outils Éditeur de liens LNK2005

pour savoir plus sur : erreur des outils Éditeur de liens LNK2005

## Erreur des outils Éditeur de liens LNK2001

pour savoir plus sur : erreur des outils Éditeur de liens LNK2001

Afficher plus ▼