Développement Linux avec C++ | Créer et déboguer des applications pour Linux

Utilisez C++ dans Visual Studio 2017 et ultérieur pour créer et déboguer des applications pour Linux.

Créer des applications C++ Linux dans Visual Studio

BIEN DÉMARRER

Télécharger, installer et configurer la charge de travail Linux

GUIDE PRATIQUE

Se connecter à votre système Linux cible

RÉFÉRENCE

Informations de référence sur ConnectionManager

Générer des projets Linux dans Visual Studio

DÉMARRAGE RAPIDE

Créer un projet Linux

Configurer un projet Linux

Déployer, exécuter et déboguer un projet Linux

RÉFÉRENCE

Informations de référence sur les pages de propriétés dans un projet Linux

Générer des projets CMake Linux dans Visual Studio

DÉMARRAGE RAPIDE

Créer et configurer un projet CMake Linux

DIDACTICIEL

Créer des projets multiplateformes C++ dans Visual Studio

Télécharger, installer et configurer la charge de travail Linux

Article • 16/06/2023

Vous pouvez utiliser l'IDE Visual Studio dans Windows pour créer, modifier et déboguer des projets C++ qui s'exécutent sur un système Linux distant, une machine virtuelle ou le Sous-système Windows pour Linux.

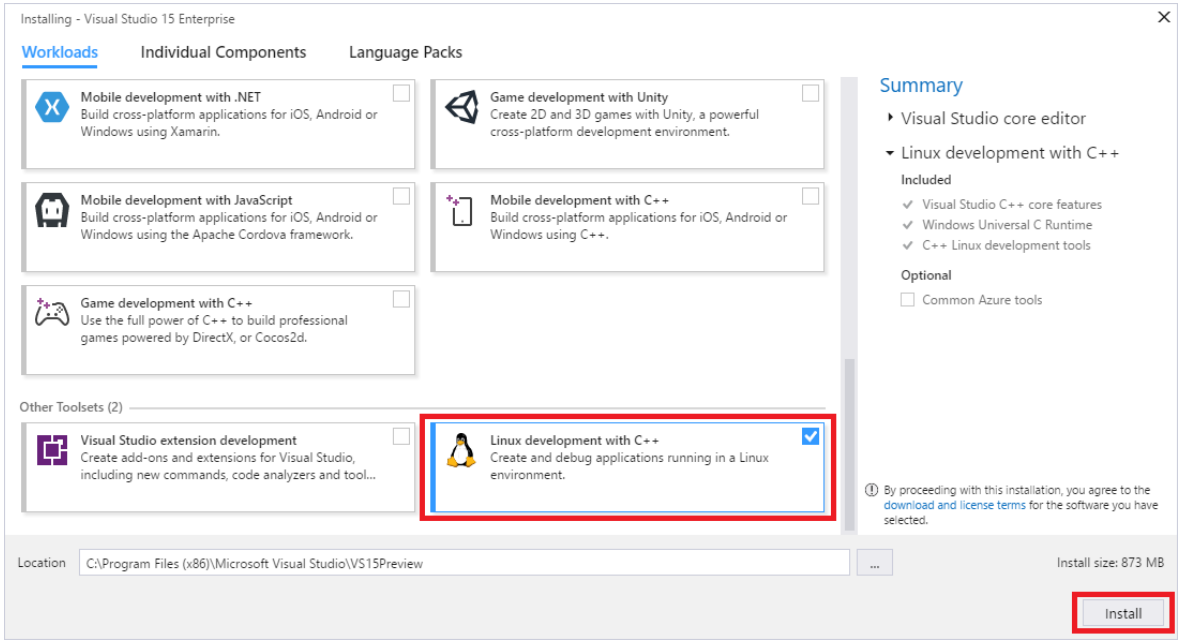
Vous pouvez utiliser votre base de code CMake existante sans avoir à la convertir en projet Visual Studio. Si votre base de code est multiplateforme, vous pouvez cibler Windows et Linux à partir de Visual Studio. Par exemple, vous pouvez modifier, générer et déboguer votre code sur Windows à l'aide de Visual Studio. Ensuite, reciblez rapidement le projet pour Linux afin de le générer et le déboguer dans un environnement Linux. Les fichiers d'en-tête Linux sont automatiquement copiés sur votre machine locale. Visual Studio les utilise pour fournir une prise en charge complète d'IntelliSense (Saisie semi-automatique des instructions, Atteindre la définition, etc.).

Tous ces scénarios nécessitent la charge de travail de **développement Linux en C++**.

Installation Visual Studio

1. Tapez « Visual Studio Installer » dans la zone de recherche Windows :

2. Recherchez le programme d'installation dans les résultats situés sous **Applications**, puis double-cliquez dessus. Lorsque le programme d'installation s'ouvre, choisissez **Modifier**, puis cliquez sur l'onglet **Charges de travail**. Faites défiler jusqu'à **Autres ensembles d'outils** et sélectionnez la charge de travail **Développement Linux avec C++**.



3. Si vous ciblez des plateformes IoT ou incorporées, accédez au volet **Détails de l'installation** à droite. Sous **Développement Linux avec C++**, développez **Composants facultatifs**, puis choisissez les composants dont vous avez besoin. La prise en charge de CMake pour Linux est sélectionnée par défaut.

4. Cliquez sur **Modifier** pour continuer l'installation.

Options pour la création d'un environnement Linux

Si vous n'avez pas encore de machine Linux, vous pouvez créer une Machine virtuelle Linux sur Azure. Pour plus d'informations, consultez Guide de démarrage rapide : Créer une machine virtuelle Linux sur le Portail Azure.

Sur Windows 10 et versions ultérieures, vous pouvez installer et cibler votre distribution Linux favorite sur le Sous-système Windows pour Linux (WSL). Pour plus d'informations, consultez Guide d'installation du sous-système Windows pour Linux pour Windows 10. Si vous ne parvenez pas à accéder au Windows Store, vous pouvez télécharger manuellement les packages de distribution WSL. WSL est un environnement de console pratique mais n'est pas recommandé pour les applications graphiques.

Les projets Linux dans Visual Studio requièrent l'installation des dépendances suivantes sur votre système Linux distant ou WSL :

- **Un compilateur** : Visual Studio 2019 et versions ultérieures ont une prise en charge complète de GCC et Clang.
- **gdb** : Visual Studio lance automatiquement gdb sur le système Linux et utilise le serveur frontal du débogueur Visual Studio pour offrir une expérience de

débogage d'une fidélité optimale sur Linux.
- **rsync** et **zip** : l'inclusion de rsync et zip permet à Visual Studio d'extraire des fichiers d'en-tête de votre système Linux vers le système de fichiers Windows pour une utilisation par IntelliSense.
- **make**
- **openssh-server** (systèmes Linux distants uniquement) : Visual Studio se connecte aux systèmes Linux distants via une connexion SSH sécurisée.
- **CMake** (projets CMake uniquement) : vous pouvez installer de fichiers binaires CMake liés statiquement pour Linux de Microsoft.
- **ninja-build** (projets CMake uniquement) : Ninja est le générateur par défaut pour les configurations Linux et WSL dans Visual Studio 2019 versions 16.6 ou ultérieures.

Les commandes suivantes supposent que vous utilisez g++ au lieu de clang.

Installation Linux : Ubuntu sur WSL

Lorsque vous ciblez WSL, il n'est pas nécessaire d'ajouter une connexion à distance ou de configurer SSH pour compiler et déboguer. **zip** et **rsync** sont nécessaires pour la synchronisation automatique des en-têtes Linux avec Visual Studio en vue de la prise en charge d'Intellisense. **ninja-build** n'est requis que pour des projets CMake. Si les applications requises sont absentes, vous pouvez les installer à l'aide de la commande suivante :

```Bash
sudo apt-get install g++ gdb make ninja-build rsync zip
```

Ubuntu sur les systèmes Linux distants

Opensh-server, **g++**, **gdb** et **make** doivent être installés sur le système Linux cible. **ninja-build** n'est requis que pour des projets CMake. Le démon **ssh** doit être en cours d'exécution. **zip** et **rsync** sont requis pour la synchronisation automatique des en-têtes distants avec votre machine locale pour la prise en charge d'Intellisense. Si ces applications sont absentes, vous pouvez les installer comme suit :

1. À l'invite de commandes du shell sur votre ordinateur Linux, exécutez :

   ```Bash
   sudo apt-get install openssh-server g++ gdb make ninja-build rsync zip
   ```

Vous pouvez être invité à indiquer votre mot de passe racine pour exécuter la commande sudo. Dans ce cas, entrez-la et continuez. Quand vous avez terminé, les outils et services obligatoires sont installés.

2. Vérifiez que le service ssh est en cours d'exécution sur votre ordinateur Linux en exécutant :

```
Bash

sudo service ssh start
```

Cette commande démarre le service qui s'exécute en arrière-plan, prêt à accepter des connexions.

Fedora sur WSL

Fedora utilise le programme d'installation du package **dnf**. Pour télécharger **g++**, **gdb**, **make**, **rsync**, **ninja-build** et **zip**, exécutez :

```
Bash

sudo dnf install gcc-g++ gdb rsync ninja-build make zip
```

zip et **rsync** sont nécessaires pour la synchronisation automatique des en-têtes Linux avec Visual Studio en vue de la prise en charge d'Intellisense. **ninja-build** n'est requis que pour des projets CMake.

Fedora sur les systèmes Linux distants

L'ordinateur cible exécutant Fedora utilise le programme d'installation de package **fnd**. Pour télécharger **openssh-server**, **g++**, **gdb**, **make**, **ninja-build**, **rsync** et **zip**, puis redémarrer le démon ssh, suivez ces instructions. **ninja-build** n'est requis que pour des projets CMake.

1. À l'invite de commandes du shell sur votre ordinateur Linux, exécutez :

```
Bash

sudo dnf install openssh-server gcc-g++ gdb ninja-build make rsync zip
```

Vous pouvez être invité à indiquer votre mot de passe racine pour exécuter la commande sudo. Dans ce cas, entrez-la et continuez. Quand vous avez terminé, les

outils et services obligatoires sont installés.

2. Vérifiez que le service ssh est en cours d'exécution sur votre ordinateur Linux en exécutant :

   ```Bash
   sudo systemctl start sshd
   ```

   Cette commande démarre le service qui s'exécute en arrière-plan, prêt à accepter des connexions.

Étapes suivantes

Vous êtes maintenant prêt à créer ou ouvrir un projet Linux et à le configurer pour qu'il s'exécute sur le système cible. Pour plus d'informations, consultez les pages suivantes :

- Créer un projet Linux MSBuild C++
- Configurer un projet CMake Linux

Connexion à votre système Linux cible dans Visual Studio

Article • 20/03/2023 • 12 minutes de lecture

Vous pouvez configurer un projet Linux pour cibler une machine virtuelle ou le sous-système Windows pour Linux (WSL). Pour un ordinateur distant, vous devez configurer une connexion à distance dans Visual Studio. Pour vous connecter à WSL, passez à la section Se connecter à WSL .

Lorsque vous utilisez une connexion à distance, Visual Studio génère des projets Linux C++ sur l'ordinateur distant. Peu importe s'il s'agit d'un ordinateur physique, d'une machine virtuelle dans le cloud ou de WSL. Pour générer le projet, Visual Studio copie le code source sur votre ordinateur Linux distant. Ensuite, le code est compilé en fonction des paramètres de Visual Studio.

> ⓘ **Notes**
>
> Depuis Visual Studio 2019 version 16.5, Visual Studio prend en charge les connexions de chiffrement conformes à la norme FIPS (Federal Information Processing Standard) 140-2 sécurisées aux systèmes Linux pour le développement à distance. Pour utiliser une connexion conforme à la norme FIPS, suivez les étapes décrites dans **Configurer le développement Linux à distance sécurisé compatible FIPS**.

Configurer le serveur SSH sur le système distant

Si `ssh` n'est pas encore configuré et en cours d'exécution sur votre système Linux, procédez comme suit pour l'installer. Les exemples de cet article utilisent Ubuntu 18.04 LTS avec le serveur OpenSSH version 7.6. Toutefois, les instructions doivent être les mêmes pour toute distribution utilisant une version modérément récente d'OpenSSH.

1. Sur le système Linux, installez et démarrez le serveur OpenSSH :

   Bash

   ```
   sudo apt install openssh-server
   sudo service ssh start
   ```

2. Si vous souhaitez que le serveur ssh démarre automatiquement au démarrage du système, activez-le à l'aide de systemctl :

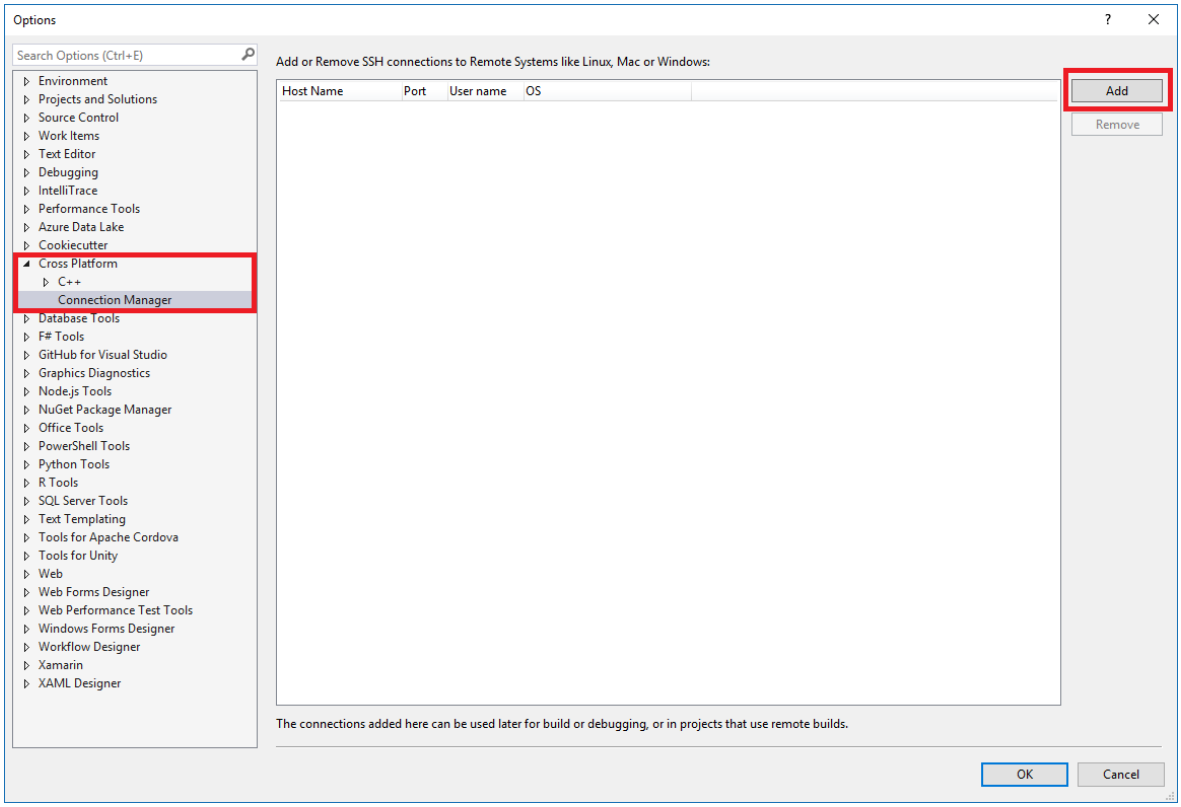
```bash
sudo systemctl enable ssh
```

Configurer la connexion à distance

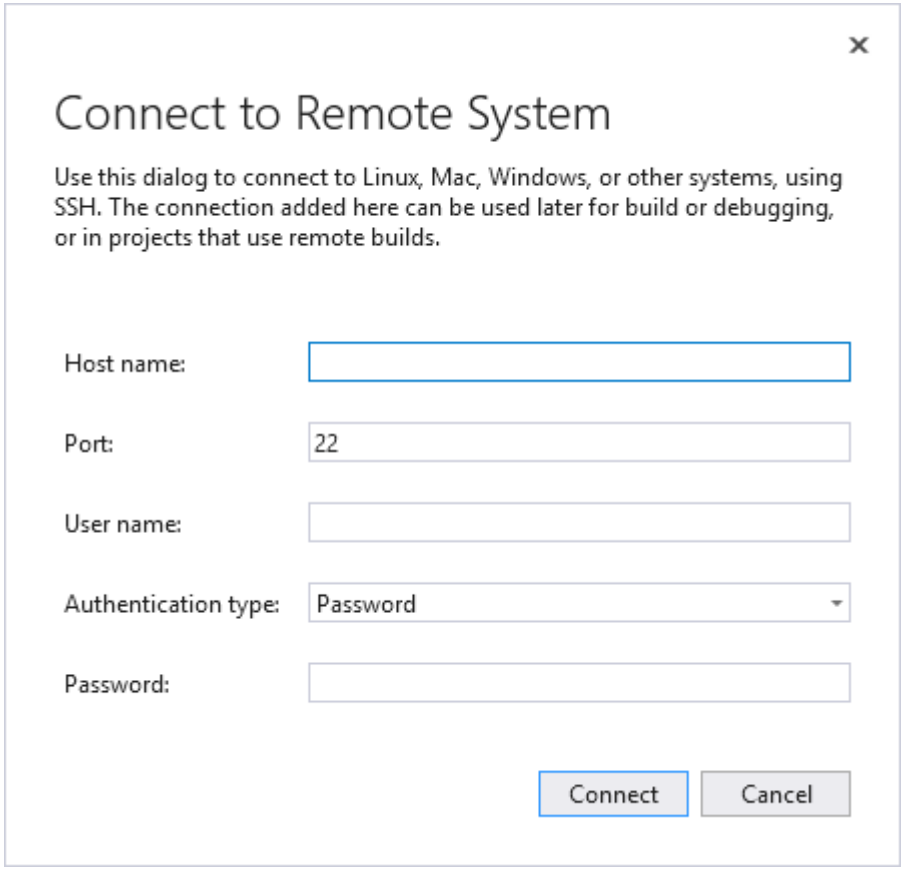
1. Dans Visual Studio, choisissez **Outils** > **Options** dans la barre de menus pour ouvrir la boîte de dialogue **Options**. Sélectionnez ensuite **Multiplateforme** > **Gestionnaire des connexions** pour ouvrir la boîte de dialogue Gestionnaire des connexions.

   Si vous n'avez pas encore configuré de connexion dans Visual Studio, lorsque vous générez votre projet pour la première fois, Visual Studio ouvre la boîte de dialogue Gestionnaire de connexions pour vous.

2. Dans la boîte de dialogue Gestionnaire des connexions, choisissez le bouton **Ajouter** pour ajouter une nouvelle connexion.



Dans les deux cas, la fenêtre **Se connecter au système distant** s'affiche.



3. Entrez les informations suivantes :

Entrée	Description
Host Name	Nom ou adresse IP de votre appareil cible
Port	Port sur lequel le service SSH est exécuté, en général 22
Nom d'utilisateur	Utilisateur sous le nom duquel s'authentifier
Type d'authentification	Un mot de passe ou une clé privée sont tous deux pris en charge
Mot de passe	Mot de passe du nom d'utilisateur entré
Fichier de clé privée	Fichier de clé privée créé pour la connexion ssh
Phrase secrète	Phrase secrète utilisée avec la clé privée sélectionnée ci-dessus

Vous pouvez utiliser un mot de passe ou un fichier de clé avec une phrase secrète pour l'authentification. Pour de nombreux scénarios de développement, l'authentification par mot de passe est suffisante, mais les fichiers clés sont plus sécurisés. Si vous avez déjà une paire de clés, il est possible de la réutiliser. Actuellement, Visual Studio prend uniquement en charge les clés RSA et DSA pour les connexions à distance.

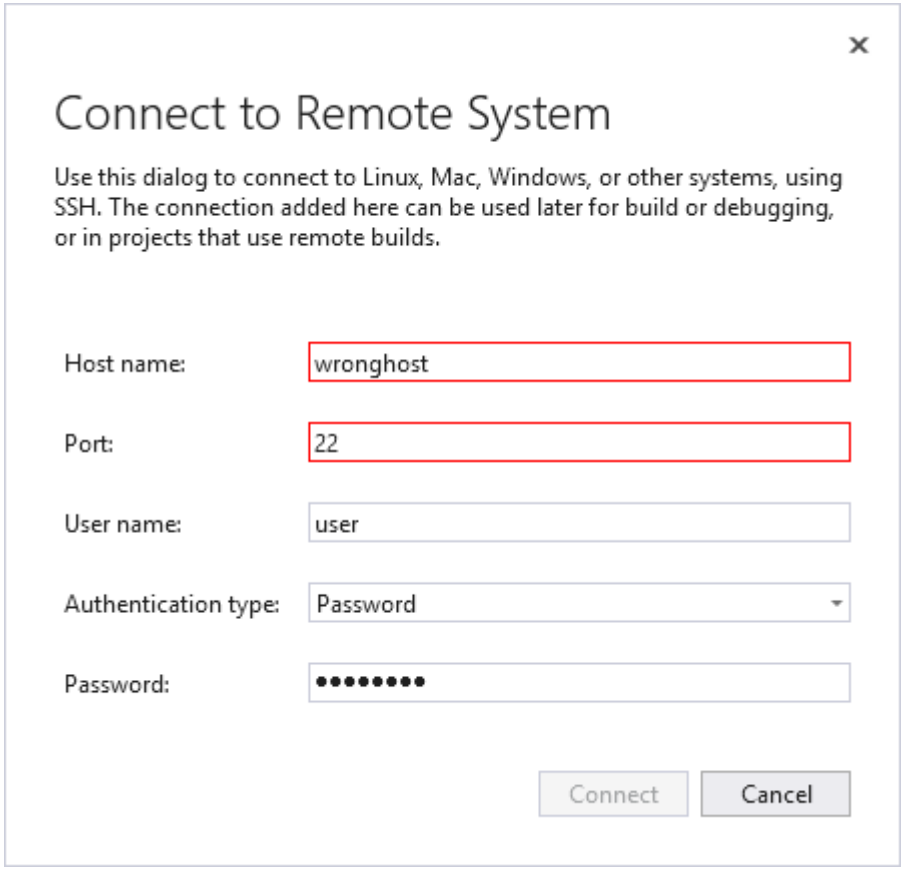
> ⓘ **Notes**

> Si vous utilisez `ssh-keygen` pour créer la clé privée, vous devez spécifier le commutateur `-m pem`, sinon la clé ne sera pas acceptée par Visual Studio. Si votre clé privée commence par `-----BEGIN OPENSSH PRIVATE KEY-----`, vous devez la convertir avec `ssh-keygen -p -f <FILE> -m pem`.

4. Cliquez sur le bouton **Se connecter** pour tenter une connexion à l'ordinateur distant.

   Si la connexion réussit, Visual Studio configure IntelliSense pour utiliser les en-têtes distants. Pour plus d'informations, consultez IntelliSense pour les en-têtes sur les systèmes distants.

   Si la connexion échoue, les zones des entrées qui doivent être modifiées seront surlignées en rouge.



   Si vous utilisez des fichiers de clé pour l'authentification, vérifiez que le serveur SSH de la machine cible est en cours d'exécution et configuré correctement.

   Si vous rencontrez des problèmes de connexion à WSL sur `localhost`, consultez Résoudre les problèmes de connexion au localhost WSL.

Vérification de la clé d'hôte

Dans Visual Studio version 16.10 ou ultérieure, vous êtes invité à vérifier l'empreinte digitale de la clé d'hôte du serveur chaque fois que Visual Studio se connecte à un système distant pour la première fois. Vous connaissez peut-être ce processus si vous avez déjà utilisé le client de ligne de commande OpenSSH ou PuTTY. L'empreinte digitale identifie le serveur. Visual Studio utilise l'empreinte digitale pour s'assurer qu'il se connecte au serveur prévu et approuvé.

La première fois que Visual Studio établit une nouvelle connexion à distance, vous êtes invité à accepter ou à refuser l'empreinte digitale de la clé d'hôte présentée par le serveur. Ou chaque fois qu'une empreinte digitale mise en cache est modifiée. Vous pouvez également vérifier une empreinte digitale à la demande en sélectionnant une connexion dans le Gestionnaire des connexions, puis en choisissant **Vérifier**.

Si vous effectuez une mise à niveau vers Visual Studio 16.10 ou version ultérieure à partir d'une version antérieure, toutes les connexions à distance existantes sont traitées comme de nouvelles connexions. Vous serez d'abord invité à accepter l'empreinte digitale de la clé d'hôte. Ensuite, Visual Studio établit une connexion et met en cache l'empreinte digitale acceptée.

Vous pouvez également mettre à jour les connexions à distance à partir de `ConnectionManager.exe` en utilisant l'argument `update`.

Algorithmes SSH pris en charge

Depuis Visual Studio version 16.9, la prise en charge des algorithmes SSH plus anciens non sécurisés utilisés pour chiffrer les données et les clés d'échange a été supprimée. Seuls les algorithmes suivants sont pris en charge. Ils sont pris en charge pour la communication SSH client à serveur et serveur à client :

Type d'algorithme	**Algorithmes pris en charge**

Chiffrement	`aes128-cbc`
	`aes128-ctr`
	`aes192-cbc`
	`aes192-ctr`
	`aes256-cbc`
	`aes256-ctr`

HMAC	`hmac-sha2-256` `hmac-sha2-512`

Type d'algorithme	Algorithmes pris en charge

Échange de clés	`diffie-hellman-group14-sha256`
	`diffie-hellman-group16-sha512`
	`diffie-hellman-group-exchange-sha256`
	`ecdh-sha2-nistp256`
	`ecdh-sha2-nistp384`
	`ecdh-sha2-nistp521`

Clé hôte	`ecdsa-sha2-nistp256`
	`ecdsa-sha2-nistp384`
	`ecdsa-sha2-nistp521`
	`ssh-dss`
	`ssh-rsa`

Configurer le serveur SSH

Tout d'abord, un peu de contexte. Vous ne pouvez pas sélectionner l'algorithme SSH à utiliser à partir de Visual Studio. Au lieu de cela, l'algorithme est déterminé lors de l'établissement de liaison initial avec le serveur SSH. Chaque côté (client et serveur) fournit une liste d'algorithmes qu'il prend en charge, puis le premier algorithme commun aux deux est sélectionné. La connexion réussit tant qu'il existe au moins un algorithme en commun entre Visual Studio et le serveur pour le chiffrement, HMAC, l'échange de clés, etc.

Le fichier de configuration Open SSH (*`sshd_config`*) ne configure pas l'algorithme à utiliser par défaut. Quand aucun algorithme n'est spécifié, le serveur SSH doit utiliser des valeurs par défaut sécurisées. Ces valeurs par défaut dépendent de la version et du fournisseur du serveur SSH. Si Visual Studio ne prend pas en charge ces valeurs par défaut, vous verrez probablement une erreur comme : « Impossible de se connecter au système distant. Aucun algorithme HMAC de client à serveur courant n'a été trouvé. » L'erreur peut également apparaître si le serveur SSH est configuré pour utiliser des algorithmes que Visual Studio ne prend pas en charge.

Le serveur SSH par défaut sur la plupart des distributions Linux modernes devrait fonctionner avec Visual Studio. Toutefois, vous exécutez peut-être un serveur SSH plus ancien configuré pour utiliser des algorithmes plus anciens non sécurisés. L'exemple suivant explique comment effectuer une mise à jour vers des versions plus sécurisées.

Dans l'exemple suivant, le serveur SSH utilise l'algorithme non sécurisé `hmac-sha1`, qui n'est pas pris en charge par Visual Studio 16.9. Si le serveur SSH utilise OpenSSH, vous pouvez modifier le fichier `/etc/ssh/sshd_config` comme indiqué ci-dessous pour activer

des algorithmes plus sécurisés. Pour les autres serveurs SSH, reportez-vous à la documentation du serveur pour savoir comment les configurer.

Tout d'abord, vérifiez que l'ensemble d'algorithmes que votre serveur utilise inclut les algorithmes pris en charge par Visual Studio. Exécutez la commande suivante sur l'ordinateur distant pour répertorier les algorithmes pris en charge par le serveur :

```
Bash

ssh -Q cipher; ssh -Q mac; ssh -Q kex; ssh -Q key
```

La commande produit une sortie telle que la suivante :

```
Bash

3des-cbc
aes128-cbc
aes192-cbc
aes256-cbc
...
ecdsa-sha2-nistp521-cert-v01@openssh.com
sk-ecdsa-sha2-nistp256-cert-v01@openssh.com
```

La sortie répertorie tous les algorithmes de chiffrement, de HMAC, d'échange de clés et de clé d'hôte pris en charge par votre serveur SSH. Si la liste n'inclut pas d'algorithmes pris en charge par Visual Studio, vous devez mettre à niveau votre serveur SSH avant de continuer.

Vous pouvez activer les algorithmes pris en charge par Visual Studio en modifiant `/etc/ssh/sshd_config` sur l'ordinateur distant. Les exemples suivants montrent comment ajouter différents types d'algorithmes à ce fichier de configuration.

Ces exemples peuvent être ajoutés n'importe où dans `/etc/ssh/sshd_config`. Assurez-vous qu'ils sont sur leurs propres lignes.

Après avoir modifié le fichier, redémarrez le serveur SSH (`sudo service ssh restart` sur Ubuntu) et réessayez de vous connecter à partir de Visual Studio.

Exemple de chiffrement

Ajouter : `Ciphers <algorithms to enable>`
Par exemple : `Ciphers aes128-cbc,aes256-cbc`

Exemple HMAC

Ajouter : `MACs <algorithms to enable>`

Par exemple : `MACs hmac-sha2-256,hmac-sha2-512`

Exemple d'échange de clés

Ajouter : `KexAlgorithms <algorithms to enable>`

Par exemple : `KexAlgorithms ecdh-sha2-nistp256,ecdh-sha2-nistp384`

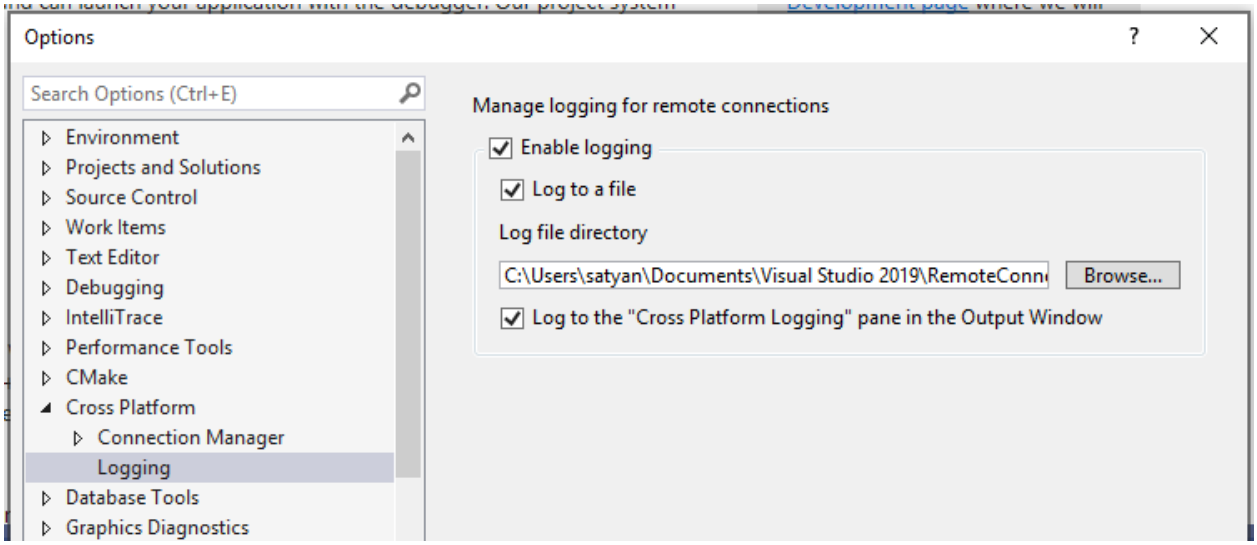
Exemple de clé d'hôte

Ajouter : `HostKeyAlgorithms <algorithms to enable>`

Par exemple : `HostKeyAlgorithms ssh-dss,ssh-rsa`

Journalisation des connexions à distance

Vous pouvez activer la journalisation pour faciliter la résolution des problèmes. Dans la barre de menus, sélectionnez **Outils>Options**. Dans la boîte de dialogue **Options**, sélectionnez **Multiplateforme** > **Journalisation** :



Les journaux incluent les connexions, toutes les commandes envoyées à la machine distante (leur texte, le code de sortie et la durée d'exécution) et toutes les sorties de Visual Studio à dans l'interpréteur de commandes. La journalisation fonctionne pour n'importe quel projet CMake multiplateforme ou projet Linux basé sur MSBuild dans Visual Studio.

Vous pouvez configurer la sortie pour qu'elle soit écrite dans un fichier ou dans le volet de **Journalisation multiplateforme** dans la fenêtre Sortie. Pour les projets Linux basés

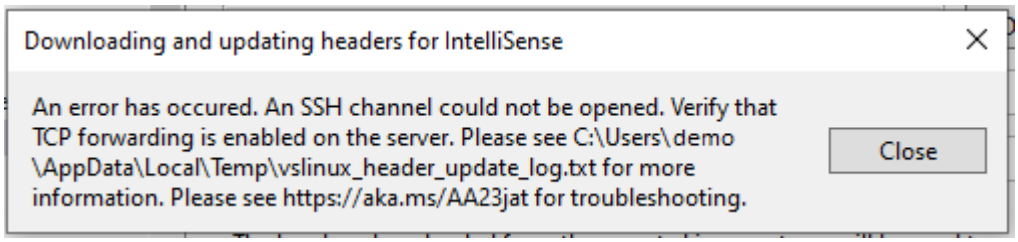
sur MSBuild, les commandes MSBuild envoyées à l'ordinateur distant ne sont pas routées vers la **Fenêtre Sortie**, car elles sont émises hors processus. Au lieu de cela, elles sont enregistrées dans un fichier avec le préfixe « msbuild_ ».

Utilitaire de ligne de commande pour le Gestionnaire des connexions

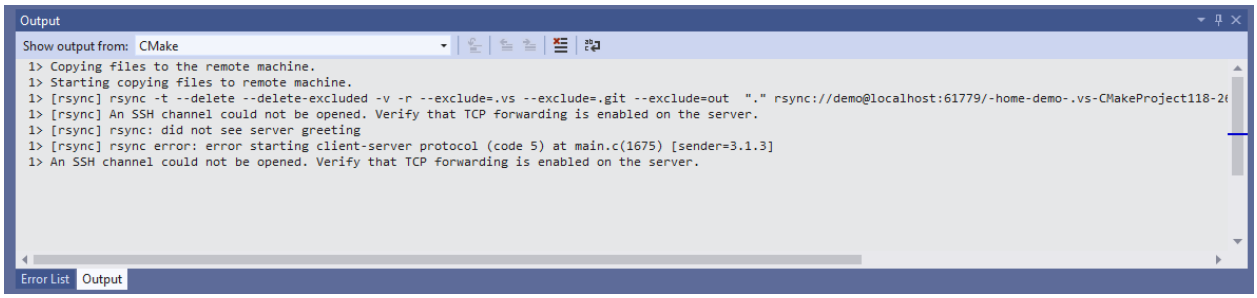
Visual Studio 2019 version 16.5 ou ultérieure : `ConnectionManager.exe` est un utilitaire de ligne de commande pour gérer les connexions de développement à distance en dehors de Visual Studio. Il est utile pour des tâches telles que l'approvisionnement d'un nouvel ordinateur de développement. Vous pouvez également l'utiliser pour configurer Visual Studio pour l'intégration continue. Pour obtenir des exemples et une référence complète à la commande ConnectionManager, consultez Référence ConnectionManager.

Transfert de port TCP

La commande `rsync` est utilisée par les projets Linux MSBuild et les projets CMake pour copier des en-têtes de votre système distant vers Windows pour une utilisation par IntelliSense. Lorsque vous ne pouvez pas activer le transfert de port TCP, désactivez le téléchargement automatique des en-têtes distants. Pour ce faire, utilisez **Outils** > **Options** > **Multiplateforme** > **Gestionnaire des connexions** > **Gestionnaire IntelliSense des en-têtes distants**. Si le transfert de port TCP n'est pas activé sur le système distant, cette erreur s'affiche lorsque le téléchargement des en-têtes distants pour IntelliSense commence :



`rsync` est également utilisé par le support CMake de Visual Studio pour copier les fichiers sources sur le système distant. Si vous ne pouvez pas activer le transfert de port TCP, vous pouvez utiliser `sftp` comme méthode de sources de copie à distance. `sftp` est souvent plus lent que `rsync`, mais n'a pas de dépendance sur le transfert de port TCP. Vous pouvez gérer votre méthode de sources de copie à distance avec la propriété `remoteCopySourcesMethod` dans l'Éditeur de paramètres CMake. Si le transfert de port TCP est désactivé sur votre système distant, une erreur s'affiche dans la fenêtre de sortie de CMake lors du premier appel de `rsync`.



`gdbserver` peut être utilisé pour le débogage sur des appareils incorporés. Si vous ne pouvez pas activer le transfert de port TCP, vous devez utiliser `gdb` pour tous les scénarios de débogage à distance. `gdb` est utilisé par défaut lors du débogage de projets sur un système distant.

La prise en charge linux de Visual Studio dépend du transfert de port TCP. `rsync` et `gdbserver` sont affectés si le transfert de port TCP est désactivé sur votre système distant. Si cette dépendance vous affecte, votez pour ce ticket de suggestion sur Developer Community.

Se connecter à WSL

Depuis Visual Studio 2019 version 16.1, Visual Studio prend en charge nativement l'utilisation de C++ avec le Sous-système Windows pour Linux (WSL). Cela signifie que vous pouvez générer et déboguer directement votre installation WSL locale. Vous n'avez plus besoin d'ajouter une connexion à distance ou de configurer SSH. Vous trouverez des détails sur l'installation de WSL ici.

Pour configurer votre installation WSL de manière à ce qu'elle fonctionne avec Visual Studio, vous devez installer les outils suivants : `gcc` ou `clang`, `gdb`, `make`, `ninja-build` (requis uniquement pour les projets CMake utilisant Visual Studio 2019 version 16.6 ou ultérieure), `rsync` et `zip`. Vous pouvez les installer sur des distributions qui utilisent `apt` à l'aide de cette commande, qui installe également le compilateur g++ :

Bash

```
sudo apt install g++ gdb make ninja-build rsync zip
```

Résoudre les problèmes de connexion au `localhost` WSL

Lorsque vous vous connectez à Sous-système Windows pour Linux (WSL) sur `localhost`, vous pouvez rencontrer un conflit avec le client Windows `ssh` sur le port 22. Dans WSL, modifiez le port qui `ssh` attend des requêtes de 23 en `/etc/ssh/sshd_config` :

```
Bash

Port 23
```

Si vous vous connectez en utilisant un mot de passe, vérifiez que les éléments suivants sont définis dans `/etc/ssh/sshd_config` :

```
Bash

# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication yes
```

Après avoir apporté ces modifications, redémarrez le serveur SSH (`sudo service ssh restart` sur Ubuntu).

Ensuite, réessayez votre connexion à `localhost` à l'aide du port 23.

Pour plus d'informations, consultez Télécharger, installer et configurer la charge de travail Linux.

Pour configurer un projet MSBuild pour WSL, consultez Configurer un projet Linux. Pour configurer un projet CMake pour WSL, consultez Configurer un projet CMake Linux. Pour suivre les instructions pas à pas permettant de créer une application console simple avec WSL, consultez ce billet de blog d'introduction sur C++ avec Visual Studio 2019 et le sous-système Windows pour Linux (WSL).

Voir aussi

Configurer un projet Linux
Configurer un projet CMake Linux
Déployer, exécuter et déboguer un projet Linux
Configurer des sessions de débogage CMake

Configurer le développement Linux à distance sécurisé compatible FIPS

Article • 03/04/2023 • 6 minutes de lecture

La Publication FIPS (Federal Information Processing Standard) 140-2 est une norme du gouvernement des États-Unis applicable aux modules de chiffrement. Les implémentations de la norme sont validées par le National Institute of Standards and Technology (NIST). Windows a validé la prise en charge des modules de chiffrement conformes à la norme FIPS. Dans Visual Studio 2019 versions 16.5 et ultérieures, vous pouvez utiliser une connexion de chiffrement sécurisée compatible FIPS à votre système Linux pour le développement à distance.

Voici comment configurer une connexion sécurisée compatible FIPS entre Visual Studio et votre système Linux distant. Ce guide s'applique lorsque vous générez des projets CMake ou MSBuild Linux dans Visual Studio. Cet article est la version compatible FIPS des instructions de connexion données dans l'article Se connecter à un ordinateur Linux distant.

Préparer une connexion compatible FIPS

Une certaine préparation est requise pour utiliser une connexion SSH sécurisée par chiffrement compatible FIPS entre Visual Studio et votre système Linux distant. Pour se conformer à la norme FIPS-140-2, Visual Studio prend uniquement en charge les clés RSA.

Les exemples fournis dans cet article utilisent Ubuntu 18.04 LTS avec un serveur OpenSSH version 7.6. Toutefois, les instructions devraient être identiques pour toute distribution utilisant une version modérément récente d'OpenSSH.

Pour configurer le serveur SSH sur le système distant

1. Sur le système Linux, installez et démarrez le serveur OpenSSH :

   Bash

   ```
   sudo apt install openssh-server
   sudo service ssh start
   ```

2. Si vous souhaitez que le serveur `ssh` démarre automatiquement lorsque le système démarre, activez-le à l'aide de `systemctl` :

```Bash
sudo systemctl enable ssh
```

3. Ouvrez `/etc/ssh/sshd_config` en tant que racine. Modifiez les lignes suivantes (ou ajoutez-les, si elles n'existent pas) :

```config
Ciphers aes256-cbc,aes192-cbc,aes128-cbc,3des-cbc
HostKeyAlgorithms ssh-rsa
KexAlgorithms diffie-hellman-group-exchange-sha256,diffie-hellman-group-exchange-sha1,diffie-hellman-group14-sha1
MACs hmac-sha2-256,hmac-sha1
```

> **Notes**
>
> `ssh-rsa` est le seul algorithme de clé d'hôte compatible FIPS pris en charge par Visual Studio. Les algorithmes `aes*-ctr` sont également compatibles FIPS, mais l'implémentation dans Visual Studio n'est pas approuvée. Les algorithmes `ecdh-*` d'échange de clés sont compatibles FIPS, mais Visual Studio ne les prend pas en charge.

   Vous n'êtes pas limité à ces options. Vous pouvez configurer `ssh` pour utiliser d'autres chiffrements, algorithmes de clé d'hôte, et ainsi de suite. D'autres options de sécurité pertinentes que vous pouvez envisager sont `PermitRootLogin`, `PasswordAuthentication` et `PermitEmptyPasswords`. Pour plus d'informations, consultez la page `man` pour `sshd_config` ou l'article Configuration du serveur SSH.

4. Après avoir enregistré et fermé `sshd_config`, redémarrez le serveur ssh pour appliquer la nouvelle configuration :

```Bash
sudo service ssh restart
```

Vous allez créer une paire de clés RSA sur votre ordinateur Windows. Ensuite, vous allez copier la clé publique dans le système Linux distant que `ssh` doit utiliser.

Pour créer et utiliser un fichier de clé RSA

1. Sur la machine Windows, générez une paire de clés RSA publique/privée à l'aide de la commande suivante :

   Invite de commandes Windows

   ```
   ssh-keygen -t rsa -b 4096 -m PEM
   ```

   La commande crée une clé publique et une clé privée. Par défaut, les clés sont enregistrées dans `%USERPROFILE%\.ssh\id_rsa` et `%USERPROFILE%\\.ssh\\id_rsa.pub`. (Dans PowerShell, utilisez `$env:USERPROFILE` plutôt que la macro cmd `%USERPROFILE%`) Si vous modifiez le nom de la clé, utilisez le nom modifié dans les étapes qui suivent. Nous vous recommandons d'utiliser une phrase secrète pour renforcer la sécurité.

2. À partir de Windows, copiez la clé publique sur la machine Linux :

   Invite de commandes Windows

   ```
   scp %USERPROFILE%\.ssh\id_rsa.pub user@hostname:
   ```

3. Sur le système Linux, ajoutez la clé à la liste de clés autorisées (et vérifiez que le fichier dispose des autorisations appropriées) :

   Bash

   ```
   cat ~/id_rsa.pub >> ~/.ssh/authorized_keys
   chmod 600 ~/.ssh/authorized_keys
   ```

4. À présent, vous pouvez tester pour voir si la nouvelle clé fonctionne dans `ssh`. Utilisez-la pour vous connecter à partir de Windows :

   Invite de commandes Windows

   ```
   ssh -i %USERPROFILE%\.ssh\id_rsa user@hostname
   ```

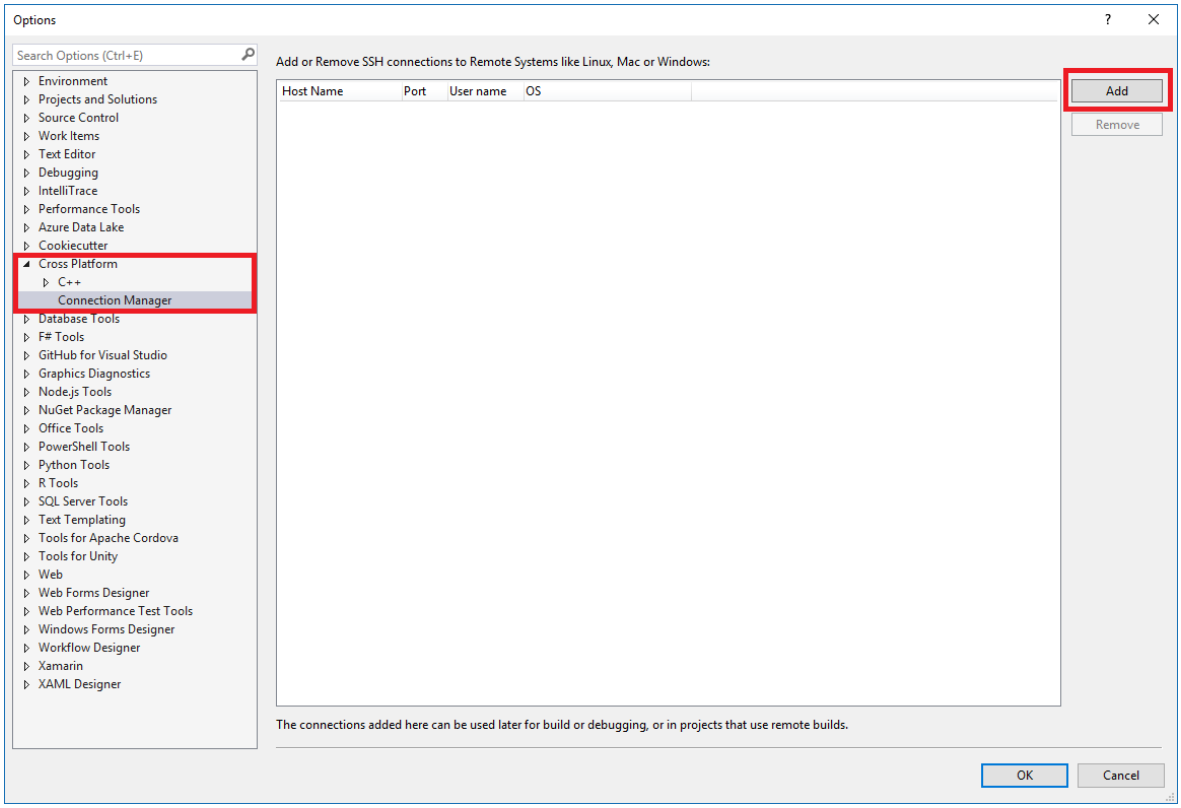
Vous avez correctement configuré `ssh`, créé et déployé des clés de chiffrement, et testé votre connexion. Vous êtes maintenant prêt à configurer la connexion Visual Studio.

Se connecter au système distant dans Visual Studio

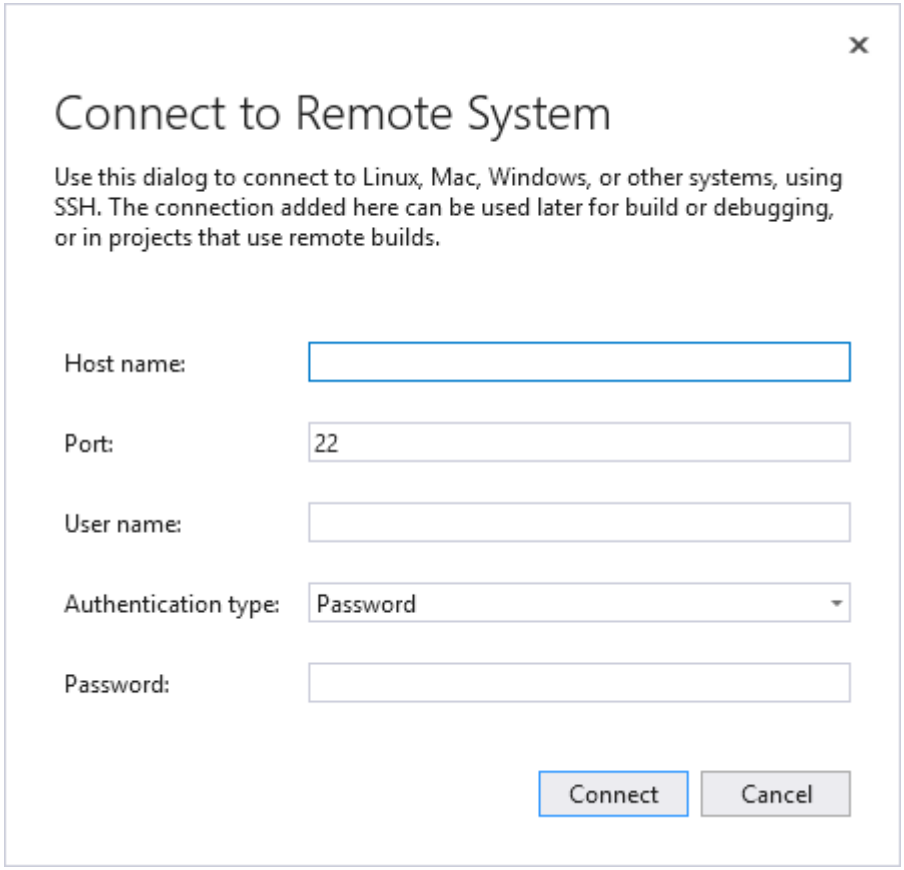
1. Dans Visual Studio, choisissez **Outils** > **Options** dans la barre de menus pour ouvrir la boîte de dialogue **Options**. Sélectionnez ensuite **Multiplateforme** > **Gestionnaire des connexions** pour ouvrir la boîte de dialogue Gestionnaire des connexions.

   Si vous n'avez pas encore configuré de connexion dans Visual Studio, lorsque vous générez votre projet pour la première fois, Visual Studio ouvre la boîte de dialogue Gestionnaire de connexions pour vous.

2. Dans la boîte de dialogue Gestionnaire des connexions, choisissez le bouton **Ajouter** pour ajouter une nouvelle connexion.



   La fenêtre **Se connecter au système distant** s'affiche.



3. Dans la boîte de dialogue **Se connecter au système distant** , entrez les détails de connexion de votre ordinateur distant.

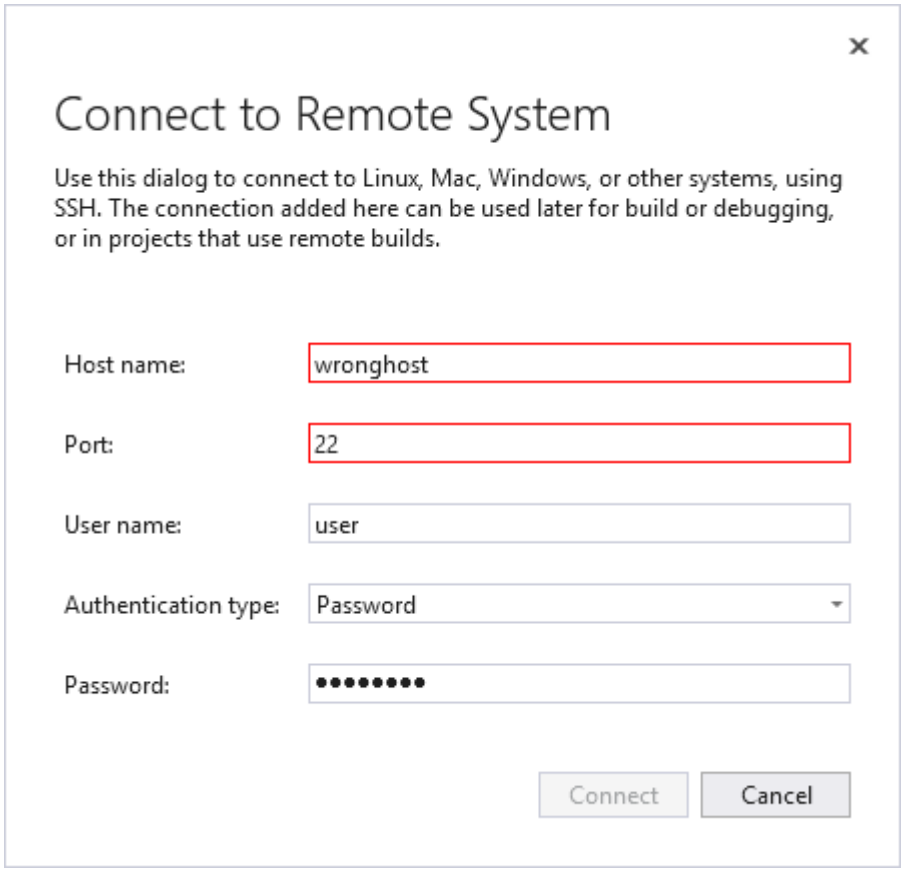
Entrée	**Description**
Host Name	Nom ou adresse IP de votre appareil cible
Port	Port sur lequel le service SSH est exécuté, en général 22
Nom d'utilisateur	Utilisateur sous le nom duquel s'authentifier
Type d'authentification	Choisir une **Clé privée** pour une connexion compatible FIPS
Fichier de clé privée	Fichier de clé privée créé pour la connexion ssh
Phrase secrète	Phrase secrète utilisée avec la clé privée sélectionnée ci-dessus

   Remplacez le type d'authentification par **clé**privée. Entrez le chemin d'accès à votre clé privée dans le champ **fichier de clé privée**. À la place, vous pouvez utiliser le bouton **Parcourir** pour accéder à votre fichier de clé privée. Ensuite, dans le champ **Phrase secrète**, entrez la phrase secrète utilisée pour chiffrer votre fichier de clé privée.

4. Cliquez sur le bouton **Se connecter** pour tenter une connexion à l'ordinateur distant.

Si la connexion réussit, Visual Studio configure IntelliSense pour utiliser les en-têtes distants. Pour plus d'informations, consultez IntelliSense pour les en-têtes sur les systèmes distants.

Si la connexion échoue, les zones des entrées qui doivent être modifiées seront surlignées en rouge.



Pour plus d'informations sur la résolution des problèmes de connexion, consultez Se connecter à votre ordinateur Linux distant.

Utilitaire de ligne de commande pour le Gestionnaire des connexions

Visual Studio 2019 version 16.5 ou ultérieure : `ConnectionManager.exe` est un utilitaire de ligne de commande pour gérer les connexions de développement à distance en dehors de Visual Studio. Il est utile pour des tâches telles que l'approvisionnement d'un nouvel ordinateur de développement. Vous pouvez également l'utiliser pour configurer Visual Studio pour l'intégration continue. Pour obtenir des exemples et une référence complète à la commande ConnectionManager, consultez Référence ConnectionManager.

Facultatif : Activer ou désactiver le mode FIPS

Il est possible d'activer le mode FIPS globalement dans Windows.

1. Pour activer le mode FIPS, appuyez sur **Windows+R** pour ouvrir la boîte de dialogue **Exécuter**, puis exécutez `gpedit.msc`.

2. Développez **stratégie d'ordinateur local** > **Configuration ordinateur** > **Paramètres Windows** > **Paramètres de sécurité** > **Stratégies locales**, puis sélectionnez **Options de sécurité**.

3. Sous **Stratégie**, sélectionnez **Chiffrement système : utilisez des algorithmes conformes aux normes FIPS pour le chiffrement, le hachage et les signatures.**, puis appuyez sur **Entrée** pour ouvrir sa boîte de dialogue.

4. Sous l'onglet **Paramètre local de sécurité**, sélectionnez **Activé** ou **Désactivé**, puis choisissez **OK** pour enregistrer vos modifications.

> ⚠ **Avertissement**
>
> L'activation du mode FIPS peut entraîner l'arrêt ou le comportement inattendu de certaines applications. Pour plus d'informations, consultez le billet de blog **Why We're Not Recommending "FIPS mode" Anymore**.

Ressources supplémentaires

Documentation Microsoft sur la validation FIPS 140

FIPS 140-2: Security Requirements for Cryptographic Modules (de NIST)

Cryptographic Algorithm Validation Program: Validation Notes (de NIST)

Billet de blog Microsoft Why We're Not Recommending "FIPS mode" Anymore

Configuration du serveur SSH

Voir aussi

Configurer un projet Linux
Configurer un projet CMake Linux
Se connecter à un ordinateur Linux distant
Déployer, exécuter et déboguer un projet Linux
Configurer des sessions de débogage CMake

Informations de référence sur ConnectionManager

Article • 16/06/2023

ConnectionManager.exe est un utilitaire de ligne de commande pour gérer les connexions de développement à distance en dehors de Visual Studio. Il est utile pour des tâches telles que l'approvisionnement d'un nouvel ordinateur de développement. Vous pouvez également l'utiliser pour configurer Visual Studio pour l'intégration continue. Vous pouvez l'utiliser dans une fenêtre d'invite de commandes développeur. Pour plus d'informations sur l'invite de commandes développeur, consultez Utiliser l'ensemble d'outils Microsoft C++ à partir de la ligne de commande.

ConnectionManager.exe est disponible dans Visual Studio 2019 version 16.5 et versions ultérieures. Il fait partie de la charge de travail de **développement Linux avec C++** dans le programme d'installation de Visual Studio. Il est également installé automatiquement lorsque vous choisissez le composant **Gestionnaire de connexions** dans le programme d'installation. Il est installé dans *%VCIDEInstallDir%\Linux\bin\ConnectionManagerExe\ConnectionManager.exe*.

La fonctionnalité de ConnectionManager.exe est également disponible dans Visual Studio. Pour gérer les connexions de développement à distance dans l'IDE, dans la barre de menus, choisissez **Outils**>**Options** pour ouvrir la boîte de dialogue Options. Dans la boîte de dialogue Options, sélectionnez **Multiplateforme**>**Gestionnaire de connexions**.

Syntaxe

> `ConnectionManager.exe` *command* [*arguments*] [*options*]

Commandes et arguments

- `add` *user@host* [`--port` *port*] [`--password` *password*] [`--privatekey` *privatekey_file*]

  Authentifie et ajoute une nouvelle connexion. Par défaut, la commande utilise le port 22 et l'authentification par mot de passe. (Vous serez invité à entrer un mot de passe.) Utilisez -`-password` et `--privatekey` pour spécifier un mot de passe pour une clé privée.

- `clean`

  Supprime le cache d'en-tête pour les connexions qui n'existent plus.

- **`help`**

  Affiche un écran d'aide.

- **`list`** [**`--properties`**]

  Affiche les informations, ID et propriétés de toutes les connexions stockées.
  Pour obtenir des exemples, consultez Propriétés couramment utilisées.

- **`modify`** [*default* | *connection_id* | *user@host* [**`--port`***port*]] [**`--property`***key=value*]

  Définit ou modifie une propriété sur une connexion.
  Si la *valeur* est vide, la clé de *propriété* est supprimée.
  En cas d'échec de l'authentification, aucune modification n'est apportée.
  Si aucune connexion n'est spécifiée (comme l'indique *default* ci-dessus), la connexion à distance par défaut de l'utilisateur est utilisée.

- **`remove`** [*connection_id* | *user@host* [**`--port`***port*]]

  Supprime une connexion. Si aucun argument n'est spécifié, vous êtes invité à spécifier la connexion à supprimer.

- **`remove-all`**

  Supprime toutes les connexions stockées.

- **`update`** [*default* | *all* | *connection_id* | *user@host* [**`--port`***port*]] [**`--previous`**] [**`--fingerprint`**]

  Ajouté dans Visual Studio 16.10. Met à jour l'empreinte digitale de la clé hôte de la ou des connexions spécifiées.

- **`version`**

  Affiche les informations de version.

Options

- **`--file`***nom_fichier*

  Lisez les informations de connexion à partir du *nom de fichier* (filname) fourni.

- **`--fingerprint`**

  Empreinte digitale de la clé hôte présentée par le serveur. Utilisez cette option **`list`** pour afficher l'empreinte digitale d'une connexion.

- **`-i`**

  Comme pour **`--privatekey`**.

- **`-n`**, **`--dry-run`**

  Effectue une exécution sèche de la commande.

- **`--no-prompt`**

  Affiche échec à la place d'une invite, le cas échéant.

- **`--no-telemetry`**

  Désactive le renvoi des données d'utilisation à Microsoft. Les données d'utilisation sont collectées et renvoyées à Microsoft, sauf si l'indicateur **`--no-telemetry`** est passé.

- **`--no-verify`**

  Ajoute ou modifie une connexion sans authentification.

- **`--p`**

  Comme pour **`--password`**.

- **`--previous`**

  Indique que la ou les connexions seront lues à partir de la version précédente du gestionnaire de connexions, mises à jour et écrites dans la nouvelle version.

- **`-q`**, **`--quiet`**

  Empêche la sortie vers `stdout` ou `stderr`.

Exemples

Cette commande ajoute une connexion pour un utilisateur nommé « user » sur localhost. La connexion utilise un fichier de clé pour l'authentification, trouvé dans *%USERPROFILE%.ssh\id_rsa*.

Invite de commandes Windows

```
ConnectionManager.exe add user@127.0.0.1 --privatekey
"%USERPROFILE%\.ssh\id_rsa"
```

Cette commande supprime la connexion qui a l'ID 1975957870 de la liste des connexions.

Invite de commandes Windows

```
ConnectionManager.exe remove 1975957870
```

Propriétés couramment utilisées

Propriété	Description
Type d'authentification	Type d'authentification utilisé pour la connexion, par exemple : `"password"`, `"privatekey"`. Pour créer une connexion avec le type d'authentification défini sur `"privatekey"` : `ConnectionManager.exe add user@127.0.0.1 --privatekey "%USERPROFILE%\.ssh\id_rsa"`
`default`	Valeur booléenne indiquant s'il s'agit de la connexion par défaut. La connexion par défaut est utilisée quand il existe plusieurs connexions disponibles et que celle à utiliser n'est pas spécifiée. Pour définir la connexion spécifiée comme connexion par défaut : `ConnectionManager.exe modify -21212121 --property default=true`
`host`	Nom ou adresse IP de l'ordinateur distant. Pour remplacer l'hôte pour la connexion spécifiée par une autre machine, en l'occurrence, un hôte local : `ConnectionManager.exe modify -21212121 --property host=127.0.0.1`
`isWsl`	Retourne true si la session à distance exécute le Sous-système Windows pour Linux.
`password`	Mot de passe pour la connexion. Modifiez le mot de passe de la connexion spécifiée avec : `ConnectionManager.exe modify -21212121 --property password="xyz"`
`platform`	Plateforme de l'ordinateur distant, telle que `"ARM"`, `"ARM64"`, `"PPC"`, `"PPC64"`, `"x64"`, `"x86"`.
`port`	Port utilisé pour la connexion. Modifiez le port pour la connexion spécifiée : `ConnectionManager.exe modify -21212121 --property port=22`

Propriété	Description
`shell`	Interpréteur de commandes préféré à utiliser sur le système distant. Les interpréteurs de commandes pris en charge sont `sh, csh, bash, tcsh, ksh, zsh, dash` Pour définir l'interpréteur de commandes préféré sur zsh pour l'ordinateur distant sur la connexion spécifiée : `ConnectionManager.exe modify -21212121 --property shell=zsh` Si l'interpréteur de commandes trouvé sur le système Linux n'est pas pris en charge, **`sh`** il est utilisé pour toutes les commandes.
`systemID`	Type de système distant, tel que `"OSX"`, `"Ubuntu"`.
`timeout`	Délai d'expiration de la connexion en millisecondes. Remplacez le délai d'expiration de la connexion spécifiée pas : `ConnectionManager.exe modify -21212121 --property timeout=100`
`username`	Nom de l'utilisateur connecté à l'ordinateur distant. Pour ajouter une connexion pour un utilisateur nommé `"user"` sur localhost : `ConnectionManager.exe add user@127.0.0.1`

Voir aussi

Connexion à votre système Linux cible dans Visual Studio

Créer un projet MSBuild C++ Linux dans Visual Studio

Article • 16/06/2023

Tout d'abord, vérifiez que la **charge de travail de développement Linux** pour Visual Studio est installée. Pour plus d'informations, consultez Télécharger, installer et configurer la charge de travail Linux.

Quand vous créez un projet C++ pour Linux dans Visual Studio, vous pouvez choisir de créer un projet Visual Studio ou un projet CMake. Cet article décrit comment créer un projet Visual Studio. En général, pour les nouveaux projets susceptibles d'inclure du code open source ou que vous envisagez de compiler pour le développement multiplateforme, nous vous recommandons d'utiliser CMake avec Visual Studio. Avec un projet CMake, vous pouvez générer et déboguer le même projet tant sur Windows que sur Linux. Pour plus d'informations, consultez Créer et configurer un projet CMake Linux.

Si vous disposez d'une solution Windows Visual Studio existante que vous souhaitez étendre afin de compiler pour Linux, et si CMake n'est pas une option, vous pouvez ajouter un projet Visual Studio Linux à la solution Windows, ainsi qu'un projet **Éléments partagés**. Placez le code partagé entre les deux plateformes dans le projet Éléments partagés, et ajoutez une référence à ce projet à partir des projets Windows et Linux.

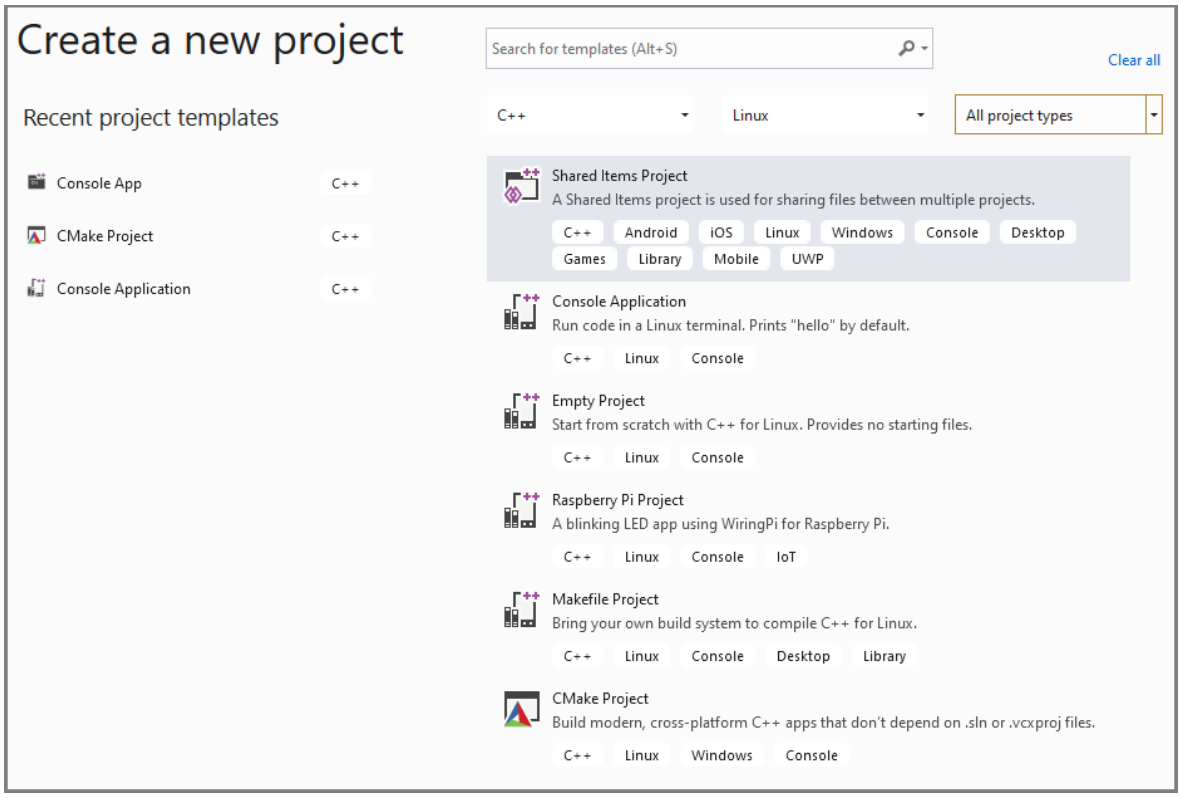
Créer un projet Linux

Pour créer un projet Linux dans Visual Studio, effectuez les étapes suivantes :

1. Sélectionnez **Fichier** > **Nouveau projet** dans Visual Studio, ou **appuyez sur Ctrl+Maj+N**. La boîte de dialogue Créer un projet s'affiche.

2. Dans la zone de texte **Rechercher des modèles**, entrez **Linux** pour afficher la liste des modèles disponibles pour les projets Linux.

3. Sélectionnez le type de projet à créer, par exemple **Application console**, puis choisissez **Suivant**. Entrez un **Nom** et un **Emplacement**, puis choisissez **Créer**.



Type de projet	Description
Projet Raspberry Pi	Projet ciblé pour un appareil Raspberry Pi, avec un exemple de code qui fait clignoter une LED
Application console	Projet ciblé pour n'importe quel ordinateur Linux, avec un exemple de code qui fait sortir un texte vers la console
Projet vide	Projet ciblé pour n'importe quel ordinateur Linux sans exemple de code
Projet Makefile	Projet ciblé pour n'importe quel ordinateur Linux, généré à l'aide d'un système de génération Makefile standard
Projet CMake	Projet ciblé pour tout ordinateur Linux, généré à l'aide du système de génération CMake

Étapes suivantes

Configurer un projet Linux MSBuild

Configurer un projet Linux MSBuild C++ dans Visual Studio

Article • 16/06/2023

Cette rubrique explique comment configurer un projet Linux basé sur MSBuild comme décrit dans Créer un projet Linux MSBuild C++ dans Visual Studio. Pour les projets Linux CMake, consultez Configurer un projet Linux CMake.

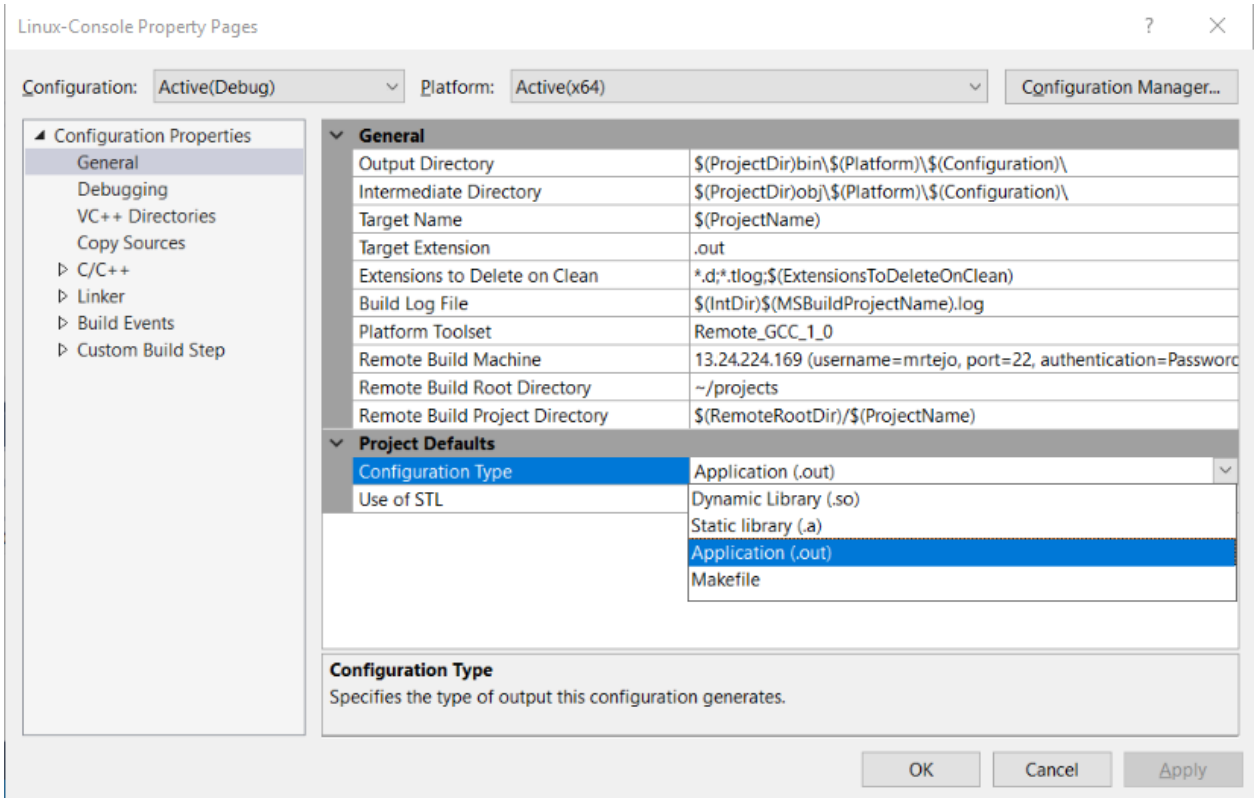
Vous pouvez configurer un projet Linux pour cibler une machine Linux physique, une machine virtuelle ou le sous-système Windows pour Linux (WSL).

Visual Studio 2019 16.1 et versions ultérieures :

- Lorsque vous ciblez WSL, vous pouvez éviter les opérations de copie nécessaires à la génération et à l'obtention d'IntelliSense requises lorsque vous ciblez un système Linux distant.
- Vous pouvez spécifier des cibles Linux distinctes pour la génération et le débogage.

Paramètres généraux :

Pour afficher les options de configuration, sélectionnez le menu **Projet** > **Propriétés** ou cliquez avec le bouton droit sur le projet dans l'**Explorateur de solutions** et sélectionnez **Propriétés** dans le menu contextuel. Les paramètres généraux s'affichent dans la section **Général**.



Par défaut, un fichier exécutable (.out) est généré. Pour générer une bibliothèque statique ou dynamique, ou utiliser un fichier makefile existant, utilisez le paramètre **Type de configuration**.

Si vous générez pour Windows Subsystem for Linux (WSL), WSL Version 1 est limité à 64 processus de compilation parallèles. Cela est régi par le paramètre **Nombre maximal de travaux de compilation parallèles** dans **Propriétés de configuration** > **C/C++** > **Général**.

Quelle que soit la version de WSL que vous utilisez, si vous envisagez d'utiliser plus de 64 processus de compilation parallèles, nous vous recommandons de générer avec Ninja qui sera généralement plus rapide et plus fiable. Pour générer avec Ninja, utilisez le paramètre **Activer la génération incrémentielle** dans **Propriétés de configuration** > **Général**.

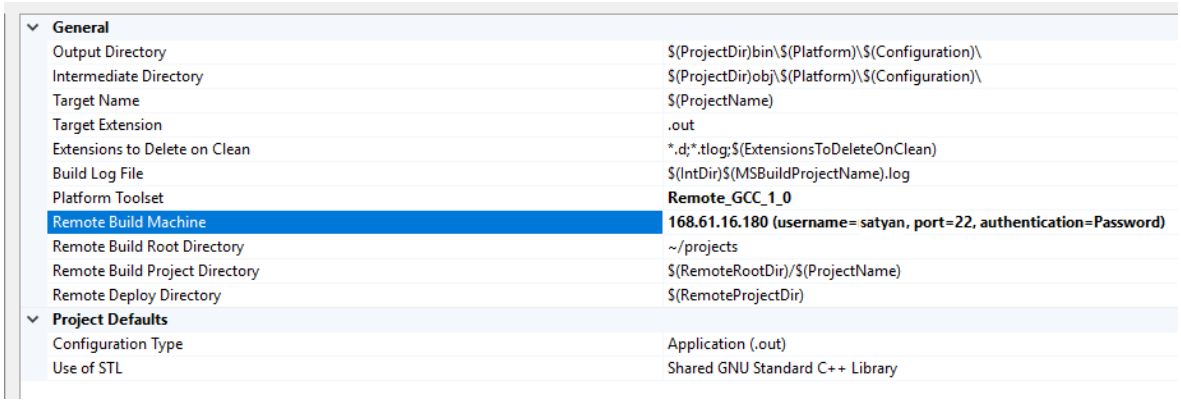
Pour plus d'informations sur les paramètres dans les pages de propriétés, consultez Informations de référence sur les pages de propriétés dans un projet Linux.

Paramètres distants

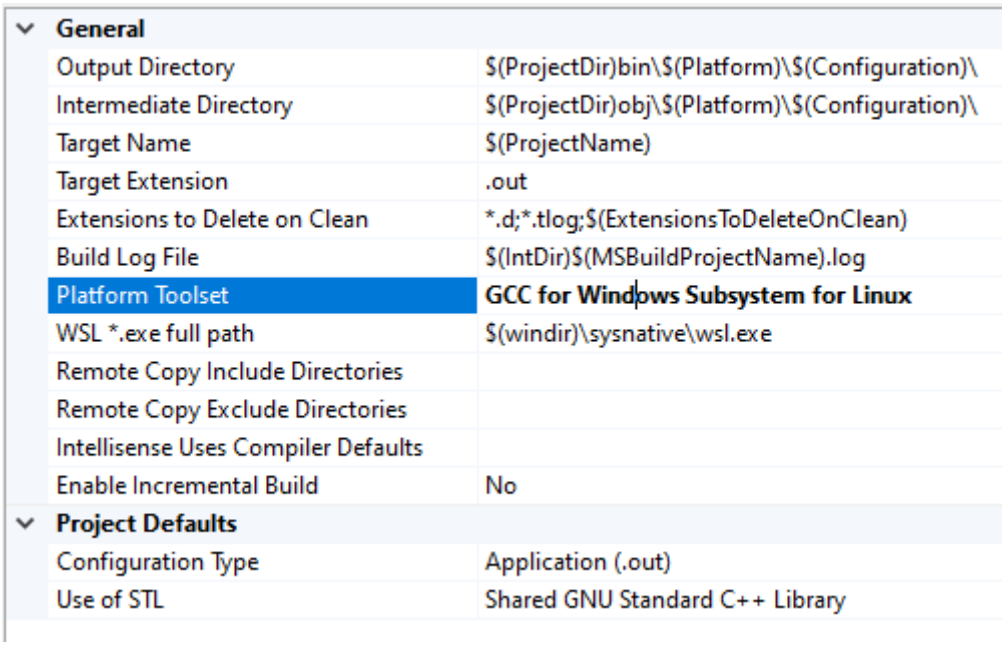
Pour changer les paramètres relatifs à l'ordinateur Linux distant, configurez les paramètres distants affichés sous Général.

- Pour spécifier un ordinateur Linux cible distant, utilisez l'entrée **Machine de build distante**. Cela vous permet de sélectionner l'une des connexions créées

précédemment. Pour créer une entrée, consultez la section Connexion à votre ordinateur Linux distant.



Visual Studio 2019 version 16.7 et versions ultérieures : pour cibler Windows Subsystem for Linux (WSL), définissez la liste déroulante **Ensemble d'outils de plateforme** sur **GCC pour Sous-système Windows pour Linux**. Les autres options distantes disparaissent et le chemin de l'interpréteur de commandes WSL par défaut s'affiche à leur place :



Si vous avez des installations de WSL côte à côte, vous pouvez spécifier ici un autre chemin. Pour plus d'informations sur la gestion de plusieurs distributions, consultez Gérer et configurer le sous-système Windows pour Linux.

Vous pouvez spécifier une autre cible pour le débogage dans la page **Propriétés de configuration**>**Débogage**.

- L'entrée **Répertoire racine de build distant** définit l'emplacement racine de l'endroit où le projet est généré sur l'ordinateur Linux distant. Par défaut et en l'absence de modification, il s'agit de **~/projects**.

- L'entrée **Répertoire de projet de build distant** définit l'emplacement où ce projet spécifique est généré sur l'ordinateur Linux distant. Par défaut, il s'agit de **$(RemoteRootDir)/$(ProjectName)**, qui se développe jusqu'à un répertoire nommé d'après le projet actuel, sous le répertoire racine défini ci-dessus.

> ⓘ **Notes**
>
> Pour changer les compilateurs C et C++ par défaut, ou l'éditeur de liens et le programme d'archivage utilisés pour générer le projet, utilisez les entrées appropriées dans les sections **C/C++** > **Général** et **Éditeur de liens** > **Général**. Vous pouvez spécifier une version spécifique de GCC ou Clang, par exemple. Pour plus d'informations, consultez **C/C++, propriétés (Linux C++)** et **Éditeur de liens, propriétés (Linux C++)**.

Copier les sources (uniquement pour les systèmes distants)

> ⓘ **Notes**
>
> Cette section ne s'applique pas quand vous ciblez WSL.

Lors de la génération sur des systèmes distants, les fichiers sources sur votre PC de développement sont copiés sur l'ordinateur Linux où ils sont compilés. Par défaut, toutes les sources dans le projet Visual Studio sont copiées aux emplacements définis dans les paramètres ci-dessus. Toutefois, des sources supplémentaires peuvent également être ajoutées à la liste, ou la copie des sources peut être entièrement désactivée, ce qui est le paramètre par défaut d'un projet Makefile.

- L'option **Sources à copier** détermine quelles sources sont copiées sur l'ordinateur distant. Par défaut, **(SourcesToCopyRemotely)** est défini sur tous les fichiers de code source dans le projet, mais n'inclut pas les fichiers d'actifs/de ressources, comme les images.

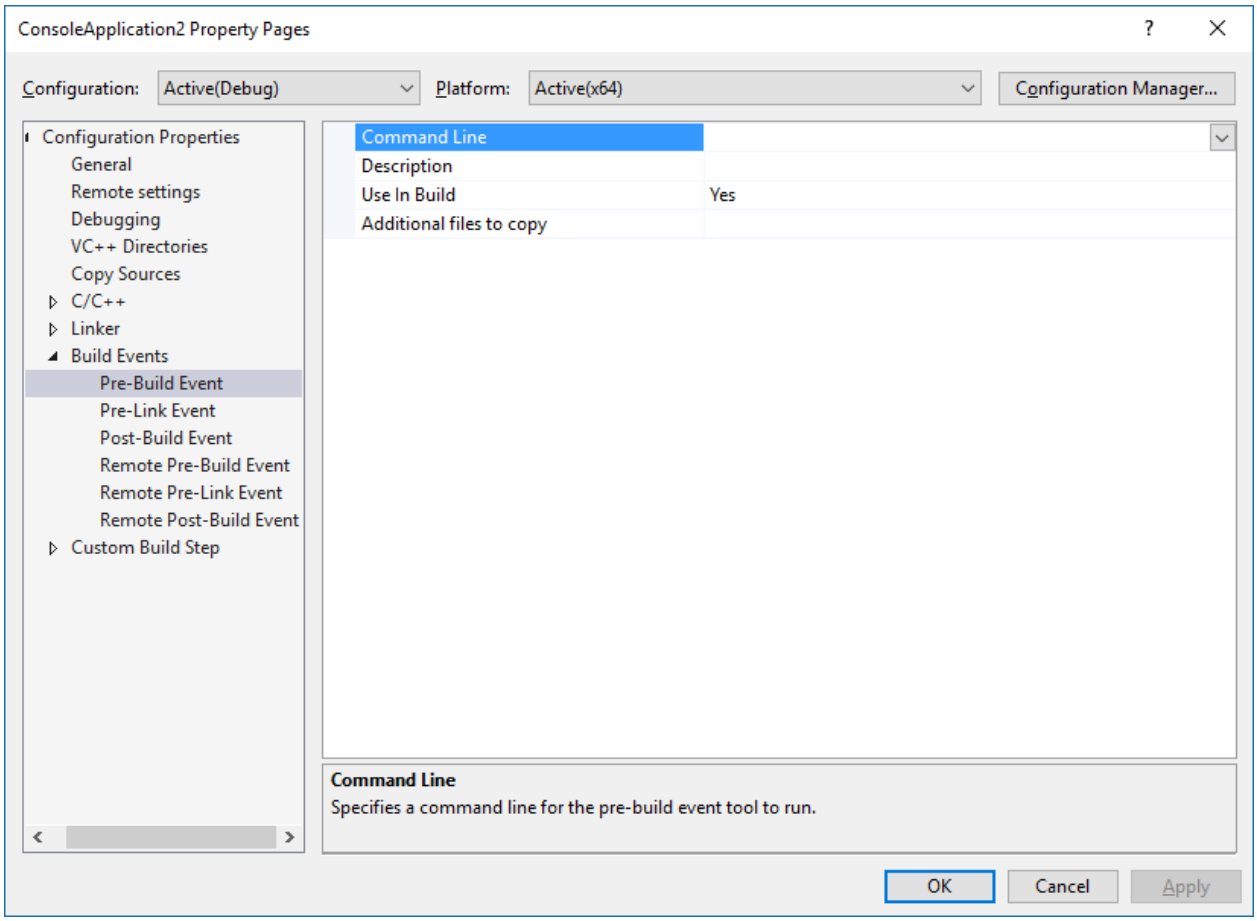
- L'option **Copier les sources** peut être activée et désactivée pour activer et désactiver la copie des fichiers sources sur l'ordinateur distant.

- L'option **Sources supplémentaires à copier** vous permet d'ajouter des fichiers sources supplémentaires qui seront copiés sur le système distant. Vous pouvez spécifier une liste délimitée par des points-virgules ou utiliser la syntaxe `:=` pour spécifier un nom local et distant à utiliser :

`C:\Projects\ConsoleApplication1\MyFile.cpp:=~/projects/ConsoleApplication1/ADifferentName.cpp;C:\Projects\ConsoleApplication1\MyFile2.cpp:=~/projects/ConsoleApplication1/ADifferentName2.cpp;`

Événements de build

Étant donné que toute la compilation se produit sur un ordinateur distant (ou WSL), plusieurs événements de build supplémentaires ont été ajoutés à la section Événements de build dans les propriétés de projet. Il s'agit des événements suivants : **Événement prébuild distant**, **Événement de préédition des liens distant** et **Événement post-build distant**. Ils se produisent sur l'ordinateur distant avant ou après les différentes étapes du processus.



IntelliSense pour les en-têtes sur les systèmes distants

Quand vous ajoutez une nouvelle connexion dans le **Gestionnaire de connexions**, Visual Studio détecte automatiquement les répertoires include pour le compilateur sur le système distant. Visual Studio compresse ensuite ces fichiers et les copie dans un répertoire sur votre ordinateur Windows local. Après cela, chaque fois que vous utilisez

cette connexion dans un projet Visual Studio ou CMake, les en-têtes dans ces répertoires sont utilisés pour fournir IntelliSense.

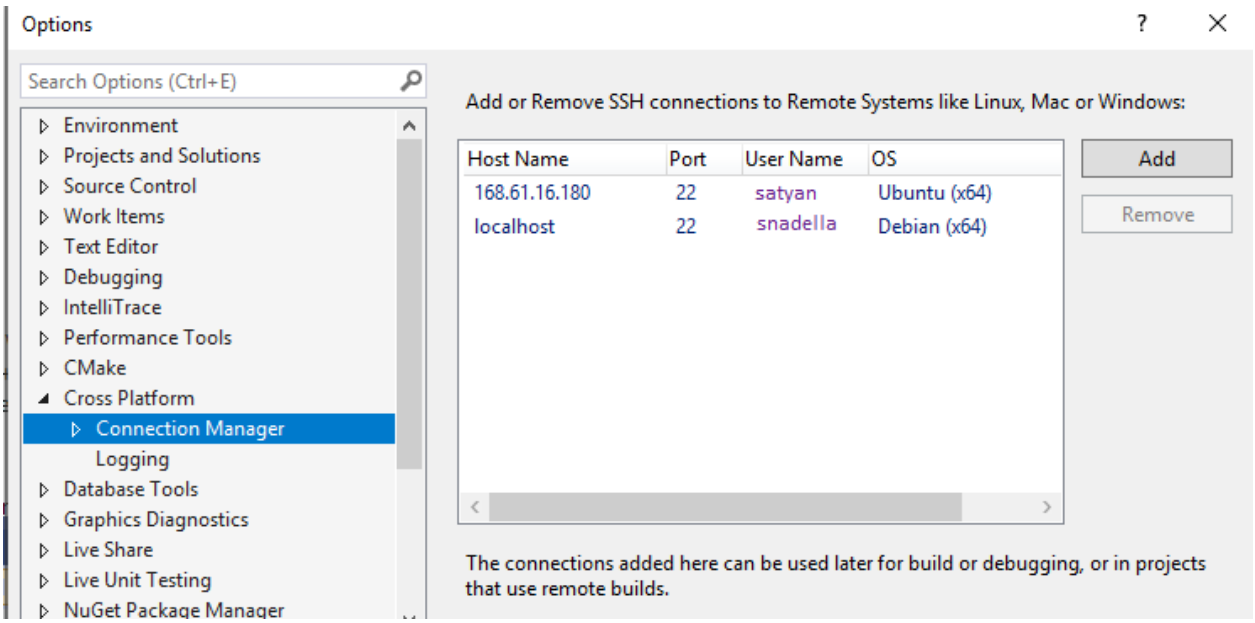
> ⓘ **Notes**
>
> Dans Visual Studio 2019 version 16.5 et ultérieure, la copie d'en-tête distante a été optimisée. Les en-têtes sont maintenant copiés à la demande lors de l'ouverture d'un projet Linux ou de la configuration de CMake pour une cible Linux. La copie se produit en arrière-plan par projet, en fonction des compilateurs spécifiés du projet. Pour plus d'informations, consultez **Améliorations apportées à la précision et aux performances de Linux IntelliSense**.

Cette fonctionnalité nécessite l'installation de zip sur l'ordinateur Linux. Pour installer zip, utilisez cette commande apt-get :

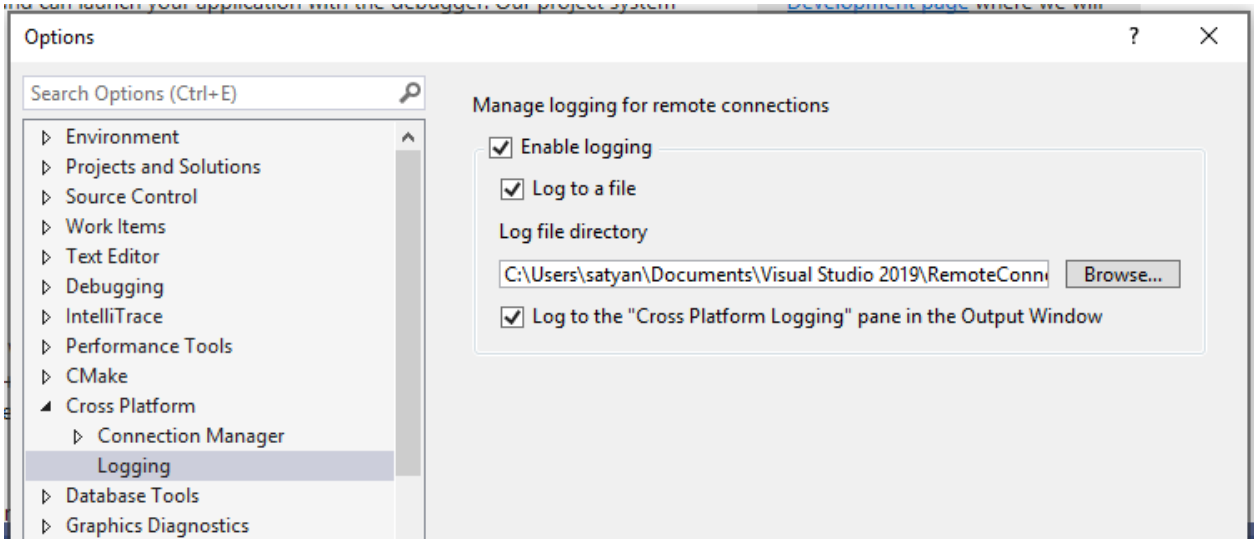
Invite de commandes Windows

```
sudo apt install zip
```

Pour gérer votre cache d'en-têtes, accédez à **Outils >Options, Multiplateforme > Gestionnaire de connexions > Gestionnaire IntelliSense des en-têtes distants**. Pour mettre à jour le cache d'en-têtes après avoir effectué des changements sur votre ordinateur Linux, sélectionnez la connexion à distance, puis sélectionnez **Mettre à jour**. Sélectionnez **Supprimer** pour supprimer les en-têtes tout en conservant la connexion. Sélectionnez **Explorer** pour ouvrir le répertoire local dans **l'Explorateur de fichiers**. Traitez ce dossier en lecture seule. Pour télécharger des en-têtes pour une connexion existante créée avant Visual Studio 2017 version 15.3, sélectionnez la connexion, puis choisissez **Télécharger**.



Vous pouvez activer la journalisation pour faciliter la résolution des problèmes :



Paramètres régionaux cibles Linux

Les paramètres de langue Visual Studio ne sont pas propagés aux cibles Linux, car Visual Studio ne gère ni ne configure pas les packages installés. Les messages affichés dans la fenêtre **Sortie**, tels que les erreurs de génération, sont présentés en utilisant la langue et les paramètres régionaux de la cible Linux. Vous devez configurer vos cibles Linux pour les paramètres régionaux souhaités.

Voir aussi

Définir des propriétés de build et de compilateur
Général C++, propriétés (Linux C++)
Répertoires VC++ (Linux C++)
Copier les sources, propriétés de projet (Linux C++)
Événement de build, propriétés (Linux C++)

Déployer, exécuter et déboguer votre projet Linux MSBuild

Article • 16/06/2023

Une fois que vous avez créé un projet Linux C++ basé sur MSBuild dans Visual Studio et que vous vous y êtes connecté par le biais du Gestionnaire de connexions Linux, vous pouvez l'exécuter et le déboguer. Vous compilez, exécutez et déboguez le code sur la cible distante.

Visual Studio 2019 version 16.1 et versions ultérieures : vous pouvez cibler différents systèmes Linux pour le débogage et la génération. Par exemple, vous pouvez utiliser la compilation croisée sur x64 et déployer sur un appareil ARM lors du ciblage de scénarios IoT. Pour plus d'informations, consultez Spécifier des machines différentes pour effectuer le build et déboguer plus loin dans cet article.

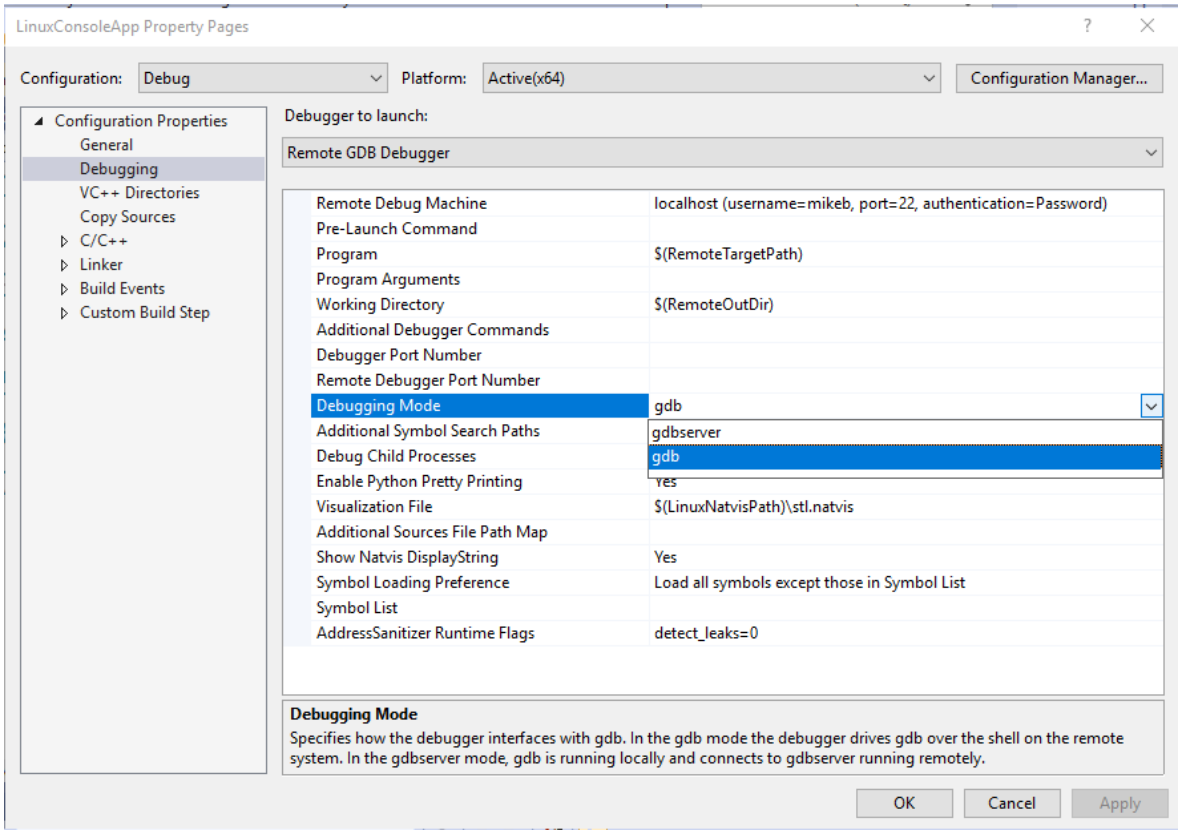
Il existe plusieurs façons de manipuler et déboguer votre projet Linux.

- Vous pouvez utiliser les fonctionnalités de débogage standard de Visual Studio, comme les points d'arrêt, les fenêtres Espion et le pointage sur une variable. Ces méthodes vous permettent de déboguer votre projet comme vous le faites habituellement pour d'autres types de projets.
- Affichez la sortie de l'ordinateur cible dans la fenêtre de console Linux. Vous pouvez également utiliser la console pour envoyer les entrées à l'ordinateur cible.

Déboguer votre projet Linux

1. Sélectionnez le mode de débogage dans la page de propriétés **Débogage**.

   GDB est utilisé pour déboguer les applications exécutées sur Linux. Lors du débogage sur un système distant (pas WSL), GDB peut s'exécuter dans deux modes différents, que vous pouvez sélectionner à partir de l'option **Mode de débogage** dans la page de propriétés **Débogage** du projet :

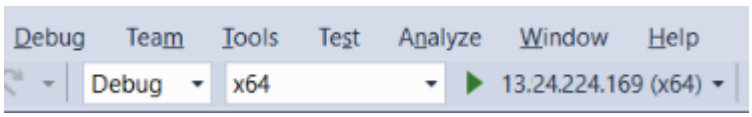


- En mode **gdbserver**, GDB s'exécute localement et se connecte à gdbserver sur le système distant.
- En mode **gdb**, le débogueur Visual Studio exécute GDB sur le système distant. C'est la meilleure option si la version locale de GDB n'est pas compatible avec la version installée sur la machine cible. Il s'agit du seul mode que la fenêtre Console Linux prend en charge.

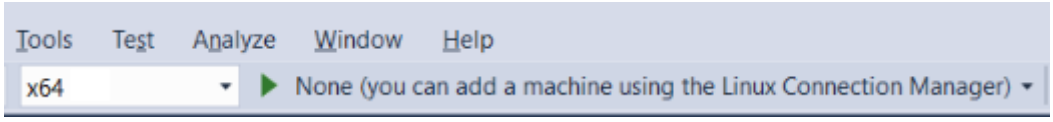
> ⓘ **Notes**
>
> Si vous ne pouvez pas atteindre les points d'arrêt en mode de débogage gdbserver, essayez le mode gdb. Vous devez d'abord **installer** gdb sur la cible distante.

2. Sélectionnez la cible distante dans la barre d'outils **Déboguer** standard de Visual Studio.

   Quand la cible distante est disponible, elle est répertoriée par son nom ou son adresse IP.



Si vous n'êtes pas encore connecté à la cible distante, vous voyez une instruction vous demandant d'utiliser le Gestionnaire de connexions Linux pour vous connecter à la cible distante.



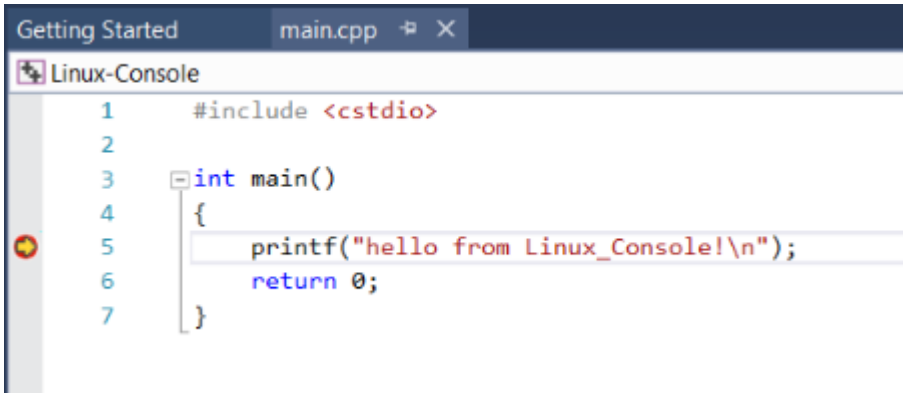
3. Définissez un point d'arrêt en cliquant dans la marge gauche d'une section de code qui s'exécute correctement.

   Un point rouge s'affiche sur la ligne de code où vous avez défini le point d'arrêt.

4. Appuyez sur **F5** (ou **Déboguer** > **Démarrer le débogage**) pour démarrer le débogage.

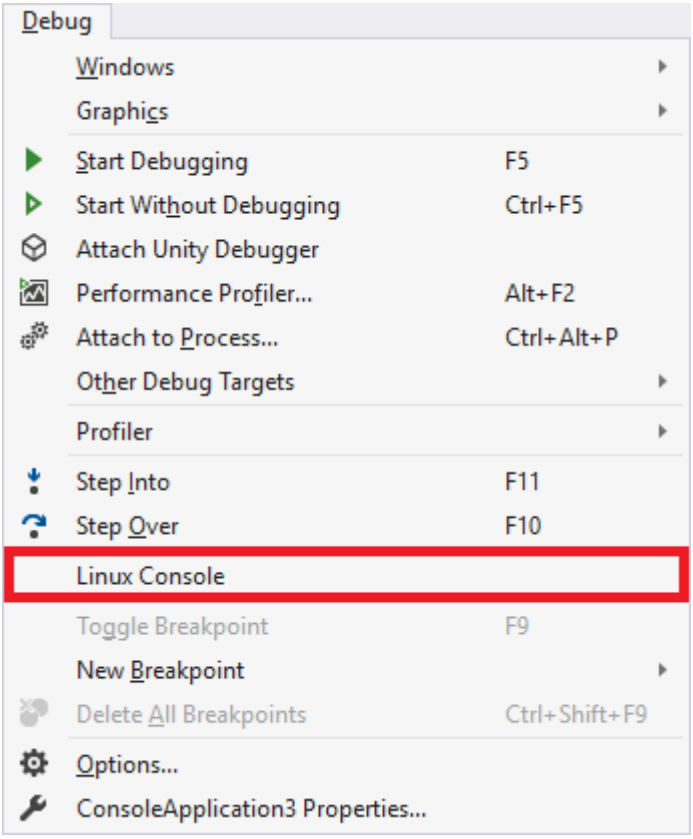
   Quand vous démarrez le débogage, l'application est compilée sur la cible distante avant de démarrer. Les erreurs de compilation éventuelles s'affichent dans la fenêtre **Liste d'erreurs**.

   S'il n'y a aucune erreur, l'application démarre et le débogueur s'interrompt au point d'arrêt.

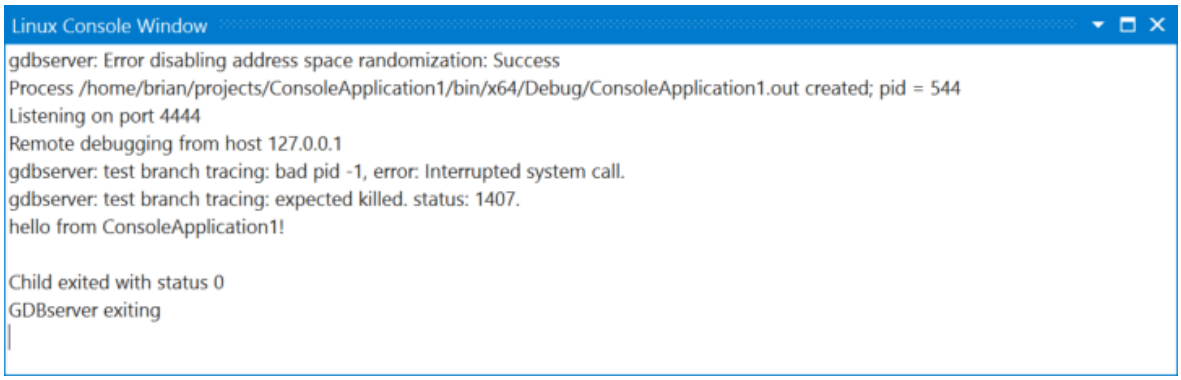


   Maintenant, vous pouvez interagir avec l'application dans son état actuel, afficher les variables et exécuter le code pas à pas en appuyant sur des touches de commande (par exemple, **F10** ou **F11**).

5. Si vous souhaitez utiliser la console Linux pour interagir avec votre application, sélectionnez **Déboguer** > **Console Linux**.



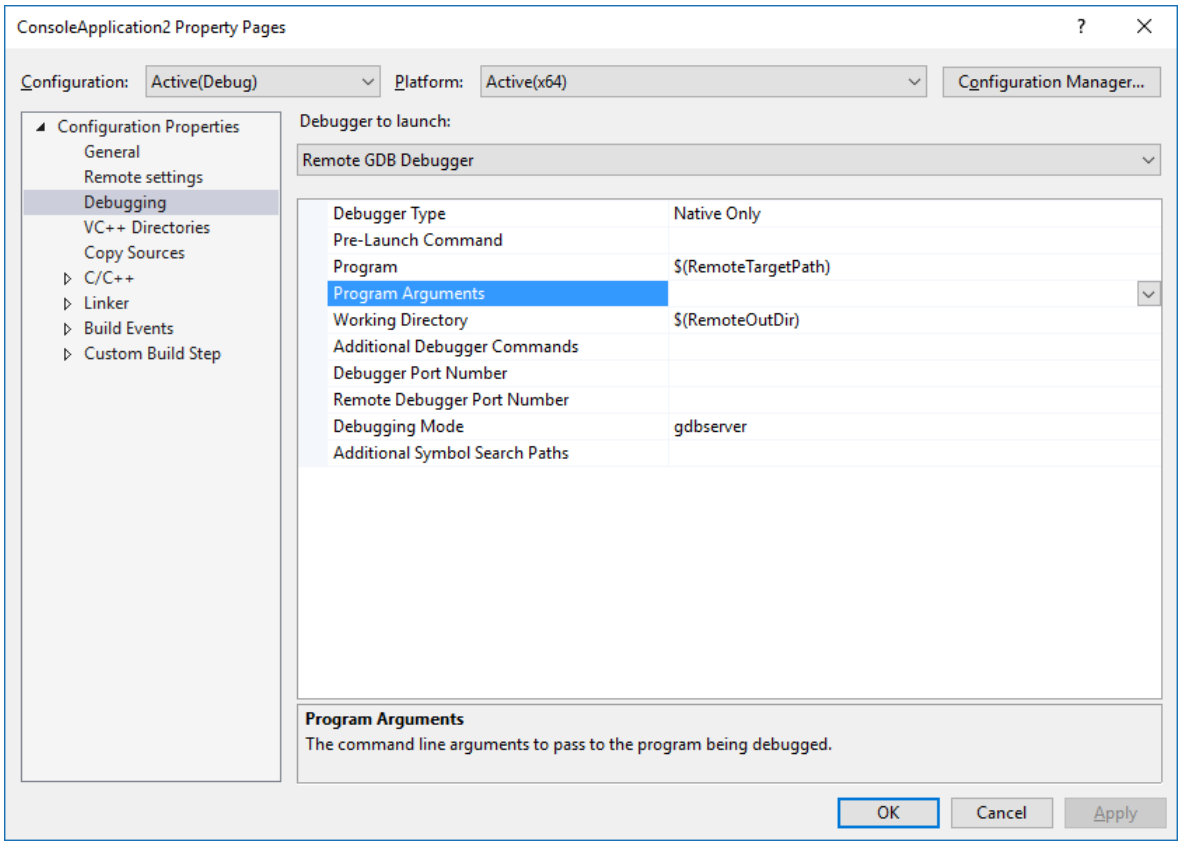
Cette console affiche toutes les sorties de console de l'ordinateur cible et prend les entrées pour les envoyer à l'ordinateur cible.



Configurer d'autres options de débogage (projets MSBuild)

- Vous pouvez passer les arguments de ligne de commande à l'exécutable en utilisant l'élément **Arguments de programme** dans la page de propriétés **Débogage** du projet.

- Vous pouvez exporter la variable d'environnement `DISPLAY` à l'aide de la **Commande de pré-lancement** dans les pages de propriétés de **Débogage** du projet. Par exemple : `export DISPLAY=:0.0`



- Vous pouvez passer des options de débogueur spécifiques à GDB à l'aide de l'entrée **Commandes de débogueur supplémentaires**. Par exemple, il est conseillé d'ignorer les signaux d'instruction non conforme (SIGILL). Vous pouvez utiliser la commande **handle** pour le faire en ajoutant ce qui suit à l'entrée **Commandes de débogueur supplémentaires** comme indiqué ci-dessus :

  `handle SIGILL nostop noprint`

- Vous pouvez spécifier le chemin d'accès à la base de données GDB utilisée par Visual Studio à l'aide de l'élément **Chemin d'accès GDB** dans la page de propriétés **Débogage** du projet. Cette propriété est disponible dans Visual Studio 2019 version 16.9 et versions ultérieures.

Déboguer avec Attacher au processus

La page de propriétés Débogage pour les projets Visual Studio et les paramètres **Launch.vs.json** pour les projets CMake ont des paramètres qui vous permettent d'effectuer un attachement à un processus en cours d'exécution. Si vous avez besoin d'un contrôle supplémentaire au-delà de ce que fournissent ces paramètres, vous pouvez placer un fichier nommé `Microsoft.MIEngine.Options.xml` à la racine de votre solution ou espace de travail. Voici un exemple simple :

XML

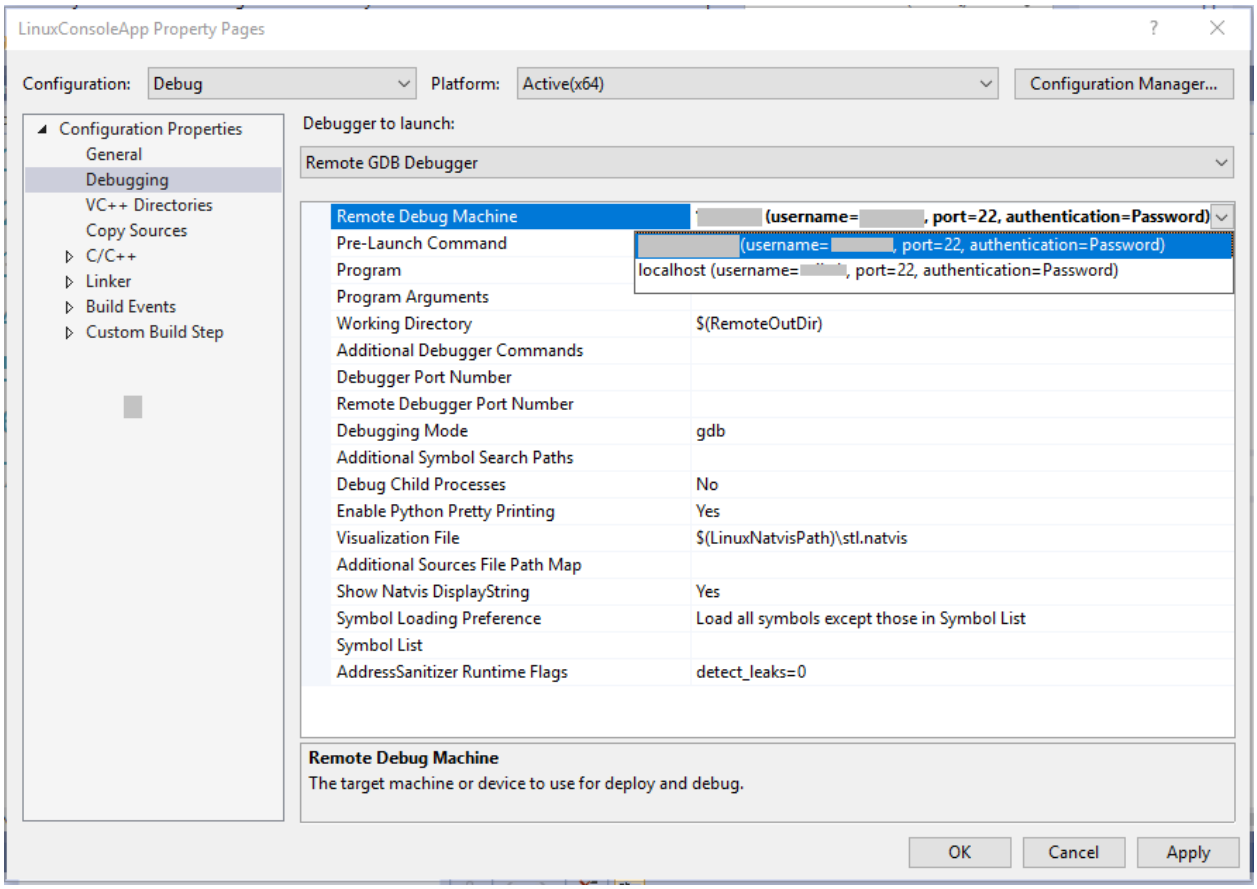
```
<?xml version="1.0" encoding="utf-8"?>
<SupplementalLaunchOptions>
    <AttachOptions>
      <AttachOptionsForConnection
AdditionalSOLibSearchPath="/home/user/solibs">
        <ServerOptions MIDebuggerPath="C:\Program Files (x86)\Microsoft
Visual Studio\Preview\Enterprise\Common7\IDE\VC\Linux\bin\gdb\7.9\x86_64-
linux-gnu-gdb.exe"
ExePath="C:\temp\ConsoleApplication17\ConsoleApplication17\bin\x64\Debug\Con
soleApplication17.out"/>
        <SetupCommands>
          <Command IgnoreFailures="true">-enable-pretty-printing</Command>
        </SetupCommands>
      </AttachOptionsForConnection>
    </AttachOptions>
</SupplementalLaunchOptions>
```

AttachOptionsForConnection contient une grande partie des attributs dont vous pouvez avoir besoin. L'exemple ci-dessus montre comment spécifier un emplacement pour rechercher des bibliothèques .so supplémentaires. L'élément enfant **ServerOptions** permet d'effectuer un attachement au processus distant avec gdbserver à la place. Pour ce faire, vous devez spécifier un client gdb local (celui fourni dans Visual Studio 2017 est indiqué ci-dessus) et une copie locale du binaire contenant les symboles. L'élément **SetupCommands** permet de transmettre des commandes directement à gdb. Vous pouvez trouver toutes les options disponibles dans le schéma LaunchOptions.xsd sur GitHub.

Spécifier différentes machines pour la génération et le débogage dans les projets Linux basés sur MSBuild

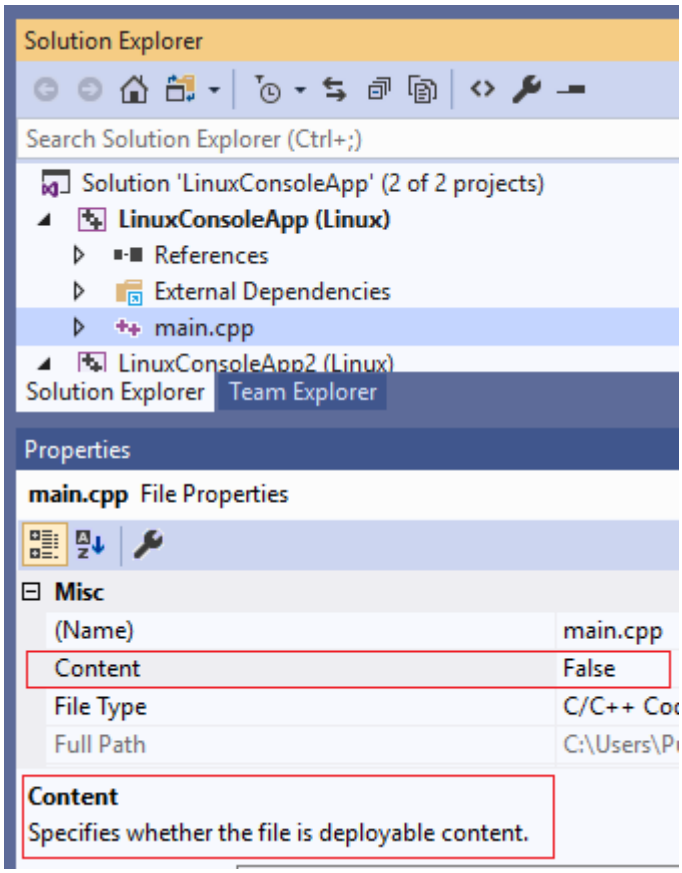
Vous pouvez séparer votre machine de build distante de votre machine de débogage distante pour les projets Linux basés sur MSBuild et les projets CMake qui ciblent une machine Linux distante. Par exemple, vous pouvez maintenant utiliser la compilation croisée sur x64 et déployer sur un appareil ARM lors du ciblage de scénarios IoT.

Par défaut, la machine de débogage distante est identique à celle de build distante (**Propriétés de configuration** > **Général** > **Machine de build distante**). Pour spécifier une nouvelle machine de débogage à distance, cliquez sur le projet dans **l'Explorateur de solutions** et accédez à **Propriétés de configuration** > **Débogage** > **Machine de débogage distante**.



Le menu déroulant pour **Machine de débogage distante** est renseigné avec toutes les connexions à distance établies. Pour ajouter une nouvelle connexion à distance, accédez à **Outils**>**Options**>**Multiplateforme**>**Gestionnaire de connexions** ou recherchez « Gestionnaire de connexions » dans **Lancement rapide**. Vous pouvez également spécifier un nouveau répertoire de déploiement distant dans les pages de propriétés du projet (**Propriétés de configuration**>**Général**>**Répertoire de déploiement distant**).

Par défaut, seuls les fichiers nécessaires pour le processus à déboguer sont déployés sur la machine de débogage distante. Vous pouvez utiliser **l'Explorateur de solutions** pour configurer les fichiers source qui seront déployés sur la machine de débogage distante. Lorsque vous cliquez sur un fichier source, vous voyez un aperçu de ses propriétés de fichier directement en dessous de l'Explorateur de solutions.



La propriété **Contenu** spécifie si le fichier doit être déployé sur la machine de débogage distante. Vous pouvez désactiver le déploiement entièrement en accédant à **Pages de propriétés**>**Gestionnaire de configuration** puis en décochant **Déployer** pour la configuration souhaitée.

Dans certains cas, vous pourriez avoir besoin de davantage de contrôle sur le déploiement de votre projet. Par exemple, si certains fichiers que vous souhaitez déployer peuvent se trouver en dehors de votre solution ou si vous souhaitez personnaliser votre répertoire de déploiement distant par fichier ou répertoire. Dans ce cas, ajoutez le ou les blocs de code suivants à votre fichier .vcxproj et remplacez « example.cpp » par les noms des fichiers que vous souhaitez utiliser :

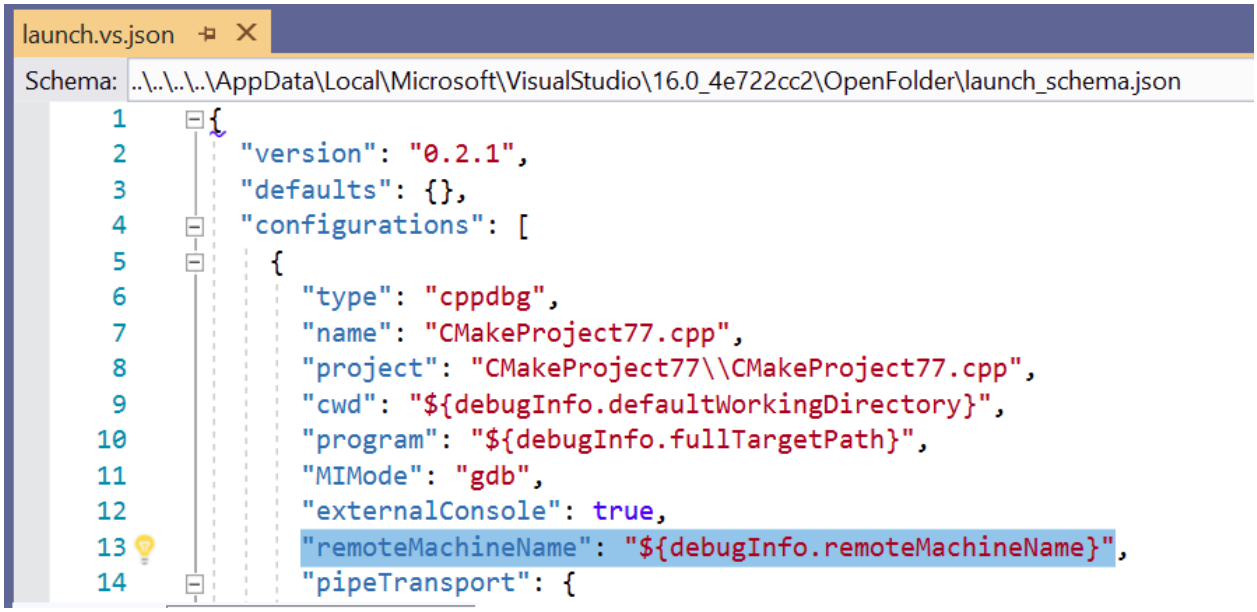
XML

```
<ItemGroup>
   <RemoteDeploy Include="__example.cpp">
<!-- This is the source Linux machine, can be empty if DeploymentType is
LocalRemote -->
      <SourceMachine>$(RemoteTarget)</SourceMachine>
      <TargetMachine>$(RemoteDebuggingTarget)</TargetMachine>
      <SourcePath>~/example.cpp</SourcePath>
      <TargetPath>~/example.cpp</TargetPath>
<!-- DeploymentType can be LocalRemote, in which case SourceMachine will be
empty and SourcePath is a local file on Windows -->
      <DeploymentType>RemoteRemote</DeploymentType>
<!-- Indicates whether the deployment contains executables -->
```

```
      <Executable>true</Executable>
    </RemoteDeploy>
</ItemGroup>
```

Projets CMake

Pour les projets CMake qui ciblent une machine Linux distante, vous pouvez spécifier une nouvelle machine de débogage distante dans launch.vs.json. Par défaut, la valeur de « remoteMachineName » est synchronisée avec la propriété « remoteMachineName » dans CMakeSettings.json, qui correspond à votre machine de build distante. Ces propriétés n'ont plus besoin de correspondre, et la valeur de « remoteMachineName » dans launch.vs.json détermine la machine distante utilisée pour le déploiement et le débogage.



IntelliSense propose une liste de toutes les connexions à distance établies. Vous pouvez ajouter une nouvelle connexion à distance, accédez à **Outils**>**Options**>**Multiplateforme**>**Gestionnaire de connexions** ou recherchez « Gestionnaire de connexions » dans **Lancement rapide**.

Si vous souhaitez avoir le contrôle total sur votre déploiement, vous pouvez ajouter le ou les blocs de code suivants au fichier launch.vs.json. N'oubliez pas de remplacer les valeurs d'espace réservé par de vraies valeurs :

JSON

```
"disableDeploy": false,
"deployDirectory": "~\foo",
"deploy" : [
   {
      "sourceMachine": "127.0.0.1 (username=example1, port=22,
authentication=Password)",
```

```
        "targetMachine": "192.0.0.1 (username=example2, port=22,
authentication=Password)",
        "sourcePath": "~/example.cpp",
        "targetPath": "~/example.cpp",
        "executable": "false"
    }
]
```

Étapes suivantes

- Pour déboguer les appareils ARM sur Linux, consultez ce billet de blog : Debugging an embedded ARM device in Visual Studio.

Voir aussi

Débogage C++, propriétés (Linux C++)

Informations de référence sur les pages de propriétés dans un projet Linux

Article • 16/06/2023

Cette section contient des informations de référence sur les pages de propriétés dans un projet Linux Visual C++.

- Général, propriétés (Linux)
- Débogage, propriétés (Linux)
- Répertoires VC++, propriétés (Linux)
- Copier les sources, propriétés (Linux)
- C/C++, propriétés (Linux)
- Éditeur de liens, propriétés (Linux)
- Événement de build, propriétés (Linux)
- Étape de build personnalisée, propriétés (Linux)
- Projet Makefile, propriétés (Linux)

Propriétés générales (Linux C++)

Article • 16/06/2023

Propriété	Description
Répertoire de sortie	Spécifie un chemin relatif vers le répertoire du fichier de sortie. Il peut inclure des variables d'environnement.
Répertoire intermédiaire	Spécifie un chemin d'accès relatif au répertoire des fichiers intermédiaires. Il peut inclure des variables d'environnement.
Nom de la cible	Spécifie le nom de fichier généré par ce projet.
Extension de la cible	Spécifie l'extension de fichier (par exemple, `.a`) générée par ce projet.
Extensions à supprimer lors du nettoyage	Spécification de caractères génériques séparés par des points-virgules pour les fichiers du répertoire intermédiaire à supprimer durant le nettoyage ou la régénération.
Fichier journal de génération	Spécifie le fichier journal de génération à utiliser quand la journalisation de la génération est activée.
Ensemble d'outils de plateforme	Spécifie l'ensemble d'outils utilisé pour générer la configuration actuelle. S'il n'est pas défini, l'ensemble d'outils par défaut est utilisé.
Chemin complet du fichier *.exe WSL	**Visual Studio 2019 version 16.1** Chemin complet vers l'exécutable WSL (Windows Subsystem for Linux) utilisé pour générer et déboguer.
Machine de build distante	Machine ou appareil cibles à utiliser pour la génération, le déploiement et le débogage à distance. Vous pouvez ajouter ou modifier une connexion de machine cible en utilisant **Outils**>**Options**>**Multiplateforme**>**Gestionnaire de connexions**. **Visual Studio 2019 version 16.1** Vous pouvez spécifier une autre machine pour le débogage dans la page Débogage.
Répertoire racine de build distant	Spécifie le chemin d'un répertoire sur la machine ou l'appareil distant.
Répertoire de projet de build distant	Spécifie le chemin d'un répertoire sur la machine ou l'appareil distant du projet.

Propriété	Description
Répertoire de déploiement à distance	**Visual Studio 2019 version 16.1** Spécifie le chemin d'accès au répertoire sur la machine ou l'appareil distants pour déployer le projet.
Activer la génération incrémentielle	**Visual Studio 2019 version 16.7** Spécifie s'il faut effectuer des générations incrémentielles à l'aide du système de génération Ninja . Les générations sont généralement plus rapides pour la plupart des projets avec ce paramètre activé.
Répertoires Include de copie à distance	**Visual Studio 2019 version 16.5** Liste de répertoires à copier de manière récursive à partir de la cible Linux. Cette propriété affecte la copie d'en-tête distant pour IntelliSense, mais pas la génération. Elle peut être utilisée quand l'option **IntelliSense utilise les valeurs par défaut du compilateur** est définie sur false. Utilisez **Répertoires Include supplémentaires** sous l'onglet Général C/C++ pour spécifier des répertoires Include supplémentaires à utiliser pour IntelliSense et la génération.
Répertoires Exclude de copie à distance	**Visual Studio 2019 version 16.5** Liste de répertoires qui ne doivent *pas* être copiés à partir de la cible Linux. En règle générale, cette propriété est utilisée pour supprimer les sous-répertoires des répertoires Include.
IntelliSense utilise les valeurs par défaut du compilateur	**Visual Studio 2019 version 16.5** Indique s'il faut interroger le compilateur référencé par ce projet pour sa liste par défaut d'emplacements Include. Ces emplacements sont automatiquement ajoutés à la liste des répertoires distants à copier. Définissez cette propriété sur false uniquement si le compilateur ne prend pas en charge les paramètres de type gcc. Les compilateurs gcc et clang prennent en charge les requêtes pour les répertoires Include (par exemple, `g++ -x c++ -E -v -std=c++11`).
Type de configuration	Spécifie le type de sortie que cette configuration génère, par exemple : **Bibliothèque dynamique (.so)**, **Bibliothèque statique (.a)**, **Application (.out)** et **Makefile**
Utilisation de STL	Spécifie la bibliothèque standard C++ à utiliser pour cette configuration, par exemple : **Bibliothèque standard C++ GNU partagée** ou **Bibliothèque standard C++ GNU statique (-static)**

Débogage C++, propriétés (Linux C++)

Article • 03/04/2023 • 2 minutes de lecture

Propriété	Description	Choices
Machine de débogage distante	**Visual Studio 2019 version 16.1** : spécifie la machine sur laquelle déboguer le programme. Peut être différente de la machine de build distante spécifiée dans la page Général. Vous pouvez ajouter ou modifier une connexion d'ordinateur cible en utilisant **Outils**>**Options**>**Multiplateforme**>**Gestionnaire des connexions**.	
Commande de prélancement	Commande exécutée dans l'interpréteur de commandes avant le démarrage du débogueur, qui peut être utilisée pour changer l'environnement de débogage.	
Programme	Chemin complet sur le système distant du programme à déboguer. S'il reste vide ou s'il n'est pas changé, il renvoie par défaut à la sortie du projet actuel.	
Arguments de programme	Arguments de ligne de commande à passer au programme en cours de débogage.	
Répertoire de travail	Répertoire de travail de l'application distante. Par défaut, il s'agit du répertoire de base de l'utilisateur.	
Commandes de débogueur supplémentaires	Commandes `gdb` supplémentaires que le débogueur doit exécuter avant le démarrage du débogage.	
Numéro de port du débogueur	Numéro de port utilisé pour la communication du débogueur avec le débogueur distant. Le port ne doit pas être en cours d'utilisation sur le système local. Cette valeur doit être positive et comprise entre 1 et 65535. Si elle n'est pas indiquée, un numéro de port libre est utilisé.	
Numéro de port du débogueur distant	Numéro de port sur lequel le serveur de débogueur distant `gdbserver` est à l'écoute sur le système distant. Le port ne doit pas être en cours d'utilisation sur le système distant. Cette valeur doit être positive et comprise entre 1 et 65535. Si elle n'est pas indiquée, un numéro de port libre (à partir du numéro 4444) est utilisé.	
Mode de débogage	Spécifie la façon dont le débogueur interagit avec `gdb`. En *mode gdb*, le débogueur exécute `gdb` via l'interpréteur de commandes sur le système distant. En *mode gdbserver*, `gdb` s'exécute localement et se connecte à `gdbserver` en cours d'exécution à distance.	**gdbserver** **gdb**

Propriété	Description	Choices
Autres chemins de recherche des symboles	Chemin de recherche supplémentaire pour les symboles de débogage (solib-search-path).	
Déboguer les processus enfants	Spécifie s'il faut activer le débogage des processus enfants.	
Activer la mise en forme Python	Permet d'afficher les valeurs d'expression selon une mise en forme élégante. Uniquement pris en charge en mode de débogage gdb.	
Fichier de visualisation	Fichier de visualisation natif par défaut (.natvis) contenant les directives de visualisation des types SLT. Les autres fichiers .natvis associés à la solution actuelle sont chargés automatiquement.	
Mappage de chemins de fichiers sources supplémentaires	Équivalences de chemins supplémentaires que le débogueur utilise pour mapper les noms de fichiers sources Windows aux noms de fichiers sources Linux. Le format est « <windows-path>=<linux-path>;... ». Un nom de fichier source sous le chemin Windows est référencé comme s'il s'agissait d'un nom de fichier situé à la même position relative sous le chemin Linux. Les fichiers qui se trouvent dans le projet local ne nécessitent pas de mappage supplémentaire.	
Chemin GDB	**Visual Studio 2019 version 16.9** : spécifie le chemin d'accès au fichier exécutable GDB à utiliser par Visual Studio.	

Répertoires VC++ (Linux C++)

Article • 16/06/2023

Propriété	Description
Répertoires Include	Chemin à utiliser pour rechercher les fichiers Include durant la génération d'un projet VC++. Correspond à la variable d'environnement INCLUDE.
Répertoires de bibliothèques	Chemin à utiliser pour rechercher des fichiers bibliothèques durant la génération d'un projet VC++. Correspond à la variable d'environnement LIB.
Répertoires sources	Chemin à utiliser durant la recherche de fichiers sources à utiliser pour IntelliSense.
Exclure des répertoires	Chemin à ignorer durant la recherche de dépendances.

Copier les sources, propriétés de projet (Linux C++)

Article • 16/06/2023

Les propriétés définies dans cette page de propriétés s'appliquent à tous les fichiers contenus dans le projet, à l'exception de ceux pour lesquels des propriétés de niveau fichier sont définies.

Propriété	Description
Sources à copier	Spécifie les sources devant être copiées sur le système distant. Tout changement apporté à cette liste peut entraîner un décalage ou des répercussions dans la structure des répertoires où sont copiés les fichiers sur le système distant.
Copier les sources	Spécifie si les sources doivent être copiées sur le système distant.
Sources supplémentaires à copier	Spécifie les sources supplémentaires à copier sur le système distant. Spécifiez éventuellement des paires de mappage entre l'emplacement local et l'emplacement distant en utilisant la syntaxe suivante : fulllocalpath1:=fullremotepath1;fulllocalpath2:=fullremotepath2, où un fichier local peut être copié vers l'emplacement distant spécifié sur le système distant.

@SourcesToCopyRemotely et @DataFilesToCopyRemotely reportez-vous aux groupes d'éléments dans le fichier projet. Pour modifier les sources ou les fichiers de données copiés à distance, modifiez le fichier *vcxproj* comme suit :

XML

```
<ItemGroup>
   <MyItems Include="foo.txt" />
   <MyItems Include="bar.txt" />
   <DataFilesToCopyRemotely Include="@(MyItems)" />
</ItemGroup>
```

C/C++, propriétés (Linux C++)

Article • 16/06/2023

Général

Propriété	Description	Choices
Autres répertoires Include	Spécifie un ou plusieurs répertoires à ajouter au chemin include. Utilisez des points-virgules pour séparer plusieurs répertoires. (-I[path]).	
Format des informations de débogage	Indique le type d'informations de débogage générées par le compilateur.	**Aucune** : ne génère aucune information de débogage ; la compilation peut donc être plus rapide. **Informations de débogage minimales** : génère des informations de débogage minimales. **Informations de débogage complètes (DWARF2)** : générez des informations de débogage DWARF2.
Nom de fichier objet	Spécifie un nom de substitution pour le nom de fichier objet par défaut. Il peut s'agir d'un nom de fichier ou de répertoire. (-o [name]).	

Propriété	Description	Choices
Niveau d'avertissement	Sélectionne la rigueur avec laquelle le compilateur doit traiter les erreurs de code. Ajoutez d'autres indicateurs directement dans les **Options supplémentaires**. (/w, /Weverything).	**Désactiver tous les avertissements** : désactive tous les avertissements du compilateur. **Activer tous les avertissements** : active tous les avertissements, dont ceux qui sont désactivés par défaut.
Considérer les avertissements comme des erreurs	Considère tous les avertissements du compilateur comme des erreurs. Pour un nouveau projet, il peut être préférable d'utiliser /Werror dans toutes les compilations. Résolvez tous les avertissements pour avoir le moins de défauts de code difficiles à détecter possible.	
Avertissements supplémentaires C	Définit un ensemble de messages d'avertissement supplémentaires.	
Avertissements supplémentaires C++	Définit un ensemble de messages d'avertissement supplémentaires.	
Activer le mode détaillé	Quand le mode détaillé est activé, affiche des informations complémentaires facilitant le diagnostic de la build.	
Compilateur C	Spécifie le programme à appeler durant la compilation de fichiers sources C, ou le chemin du compilateur C sur le système distant.	
Compilateur C++	Spécifie le programme à appeler durant la compilation de fichiers sources C++, ou le chemin du compilateur C++ sur le système distant.	
Délai d'attente de la compilation	Délai d'attente de la compilation distante en millisecondes.	
Copier les fichiers objets	Indique s'il faut copier les fichiers objets compilés du système distant sur la machine locale.	
Nombre maximal de travaux de compilation parallèles	Nombre de processus à créer en parallèle pendant la compilation. La valeur par défaut est 1. Si vous utilisez un sous-système Windows pour Linux (WSL) version 1, la limite est 64.	

Propriété	Description	Choices
Valider l'architecture	Spécifiez s'il faut vérifier si la plateforme que le projet cible correspond au système distant.	
Activer un assainisseur d'adresse	Compilez le programme avec assainisseur d'adresse, qui est un détecteur d'erreur de mémoire rapide capable de trouver des problèmes de mémoire de runtime tels que use-after-free, et effectuer des vérifications hors limites.	

Optimisation

Propriété	Description	Choices
Optimisation	Indique le niveau d'optimisation pour l'application.	**Personnalisé** : optimisation personnalisée. **Désactivé** : désactive l'optimisation. **Réduire la taille** : optimise la taille. **Augmenter la vitesse** : optimise la vitesse. **Optimisation complète** : optimisations coûteuses.
Alias strict	Les règles d'alias les plus strictes sont utilisées. Un objet d'un type donné n'est jamais considéré comme ayant la même adresse qu'un objet d'un autre type.	
Dérouler les boucles	Déroule les boucles pour rendre l'application plus rapide en réduisant le nombre de branches exécutées, au prix d'un code de plus grande taille.	
Optimisation durant l'édition de liens	Active les optimisations inter-procédurales en permettant à l'optimiseur d'accéder aux fichiers objets dans votre application.	
Omettre le pointeur de frame	Empêche la création des pointeurs de frame sur la pile des appels.	

Propriété	Description	Choices
Aucun bloc commun	Alloue des variables globales, même non initialisées, dans la section de données du fichier objet, au lieu de les générer en tant que blocs communs.	

Préprocesseur

Propriété	Description
Définitions de préprocesseur	Définit les symboles de prétraitement pour votre fichier source. (-D)
Annuler la définition de définitions de préprocesseur	Spécifie l'annulation de la définition d'une ou de plusieurs définitions du préprocesseur. (-U [macro])
Annulation de la définition de toutes les définitions du préprocesseur	Annule la définition de toutes les valeurs de préprocesseur précédemment définies. (-undef)
Affichage des fichiers Include	Affiche la liste des fichiers include avec les résultats de la compilation. (-H)

Génération de code

Propriété	Description	Choices
PIC (Position Independent Code)	Génère le code PIC (Position Independent Code) à utiliser dans une bibliothèque partagée.	
Les statiques sont thread-safe	Génère du code supplémentaire qui permet d'utiliser les routines spécifiées dans l'ABI C++ pour l'initialisation thread-safe des statiques locales.	**Non** : désactive les statiques thread-safe. **Oui** : active les statiques thread-safe.
Optimisation à virgule flottante	Active les optimisations à virgule flottante en assouplissant la conformité à la norme IEEE-754.	
Méthodes inline masquées	Une fois activées, les copies hors ligne des méthodes inline sont déclarées `private extern`.	

Propriété	Description	Choices
Symboles masqués par défaut	Tous les symboles sont déclarés `private extern` sauf s'ils sont explicitement marqués pour l'exportation à l'aide de la macro `__attribute`.	
Activer les exceptions C++	Spécifie le modèle de gestion des exceptions utilisé par le compilateur.	**Non** : désactive la gestion des exceptions. **Oui** : active la gestion des exceptions.

Langage

Propriété	Description	Choices
Activer les informations de type au moment de l'exécution	Ajoute le code permettant de vérifier les types d'objet C++ à l'exécution (informations de type au moment de l'exécution). (frtti, fno-rtti)	
Norme du langage C	Détermine la norme du langage C.	**Par défaut** **C89** : norme du langage C89. **C99** : norme du langage C99. **C11** : norme du langage C11. **C99 (Dialecte GNU)** : norme du langage C99 (Dialecte GNU). **C11 (Dialecte GNU)** : norme du langage C11 (Dialecte GNU).

Propriété	Description	Choices
Norme du langage C++	Détermine la norme du langage C++.	**Par défaut** **C++03** : norme du langage C++03. **C++11** : norme du langage C++11. **C++14** : norme du langage C++14. **C++03 (Dialecte GNU)** : norme du langage C++03 (Dialecte GNU). **C++11 (Dialecte GNU)** : norme du langage C++11 (Dialecte GNU). **C++14 (Dialecte GNU)** : norme du langage C++14 (Dialecte GNU).

Avancé

Propriété	Description	Choices
Compiler en	Sélectionne l'option de langage de compilation pour les fichiers .c et .cpp. (-x c, -x c++)	**Par défaut** : effectue la détection d'après l'extension (.c ou .cpp). **Compiler en code C** : compile en code C. **Compiler en code C++** : compile en code C++.
Fichiers Include forcés	Spécifie un ou plusieurs fichiers include forcés (-include [name])	

Éditeur de liens, propriétés (Linux C++)

Article • 16/06/2023

Général

Propriété	Description	Choices
Fichier de sortie	L'option substitue le nom et l'emplacement par défaut du programme créé par l'éditeur de liens. (-o)	
Afficher la progression	Affiche les messages de progression de l'éditeur de liens.	
Version	L'option -version indique à l'éditeur de liens de placer un numéro de version dans l'en-tête de l'exécutable.	
Activer la sortie détaillée	L'option -verbose indique à l'éditeur de liens de générer la sortie des messages détaillés pour le débogage.	
Trace	L'option --trace indique à l'éditeur de liens de générer la sortie des fichiers d'entrée au fur et à mesure de leur traitement.	
Symboles de traces	Affiche la liste des fichiers dans lesquels un symbole apparaît. (--trace-symbol=symbol)	
Afficher le mappage	L'option --print-map indique à l'éditeur de liens de générer la sortie d'un mappage de liens.	
Signaler les références de symboles non résolus	Quand cette option est activée, les références des symboles non résolus sont signalées.	
Optimiser pour l'utilisation de la mémoire	Permet d'optimiser l'utilisation de la mémoire, en relisant les tables de symboles selon les besoins.	
Chemin de recherche de la bibliothèque partagée	Permet à l'utilisateur de remplir le chemin de recherche de la bibliothèque partagée. (-rpath-link=path)	
Répertoires de bibliothèques supplémentaires	Permet à l'utilisateur de substituer le chemin de la bibliothèque d'environnement. (-L folder).	
Éditeur de liens	Spécifie le programme à appeler durant l'édition des liens, ou le chemin de l'éditeur de liens sur le système distant.	
Délai d'attente de l'édition des liens	Délai d'attente de l'édition des liens distante, en millisecondes.	

Propriété	Description	Choices
Copier la sortie	Indique s'il faut copier le fichier de sortie de build du système distant sur la machine locale.	

Entrée

Propriété	Description	Choices
Bibliothèques par défaut spécifiques ignorées	Spécifie un ou plusieurs noms de bibliothèques par défaut à ignorer. (--exclude-libs lib,lib)	
Ignorer les bibliothèques par défaut	Ignore les bibliothèques par défaut et recherche uniquement les bibliothèques explicitement spécifiées.	
Forcer la non-définition des références de symboles	Force l'entrée du symbole dans le fichier de sortie en tant que symbole non défini. (-u symbol --undefined=symbol)	
Dépendances de bibliothèque	Cette option permet de spécifier des bibliothèques supplémentaires à ajouter à la ligne de commande de l'éditeur de liens. La bibliothèque supplémentaire est ajoutée à la fin de la ligne de commande de l'éditeur de liens, avec le préfixe 'lib' et l'extension '.a'. (-lFILE)	
Dépendances supplémentaires	Spécifie les éléments supplémentaires à ajouter à la ligne de commande de l'éditeur de liens.	

Débogage

Propriété	Description	Choices
Informations de symboles du débogueur	Informations de symboles du débogueur contenues dans le fichier de sortie.	**Inclure tout** **Omettre les informations de symboles du débogueur uniquement** **Omettre toutes les informations de symbole**
Nom de fichier de mappage	L'option Map indique à l'éditeur de liens de créer un fichier de mappage avec le nom spécifié par l'utilisateur. (-Map=)	

Avancé

Propriété	Description	Choices
Marquer les variables ReadOnly après le réadressage	Cette option marque les variables en lecture seule après le réadressage.	
Activer la liaison de fonction immédiate	Cette option marque l'objet pour une liaison de fonction immédiate.	
Ne pas exiger de pile exécutable	Cette option marque la sortie en indiquant qu'elle ne nécessite pas de pile exécutable.	
Archive complète	L'archive complète utilise l'ensemble du code des sources et des dépendances supplémentaires.	

Événement de build, propriétés (Linux C++)

Article • 16/06/2023

Événement prébuild

Propriété	Description
Ligne de commande	Spécifie une ligne de commande pour l'outil d'événement prébuild à exécuter.
Description	Spécifie une description de l'outil d'événement prébuild à afficher.
Utilisation dans la génération	Spécifie si cet événement de build est exclu de la génération pour la configuration actuelle.
Fichiers supplémentaires à copier	Spécifie les fichiers supplémentaires à copier sur le système distant. Spécifiez éventuellement des paires de mappage entre l'emplacement local et l'emplacement distant en utilisant la syntaxe suivante : fulllocalpath1:=fullremotepath1;fulllocalpath2:=fullremotepath2, où un fichier local peut être copié vers l'emplacement distant spécifié sur le système distant.

Événement de préédition des liens

Propriété	Description
Ligne de commande	Spécifie une ligne de commande pour l'outil d'événement de préédition des liens à exécuter.
Description	Spécifie une description de l'outil d'événement de préédition des liens à afficher.
Utilisation dans la génération	Spécifie si cet événement de build est exclu de la génération pour la configuration actuelle.
Fichiers supplémentaires à copier	Spécifie les fichiers supplémentaires à copier sur le système distant. Spécifiez éventuellement des paires de mappage entre l'emplacement local et l'emplacement distant en utilisant la syntaxe suivante : fulllocalpath1:=fullremotepath1;fulllocalpath2:=fullremotepath2, où un fichier local peut être copié vers l'emplacement distant spécifié sur le système distant.

Événement post-build

Propriété	Description
Ligne de commande	Spécifie une ligne de commande pour l'outil d'événement postbuild à exécuter.
Description	Spécifie une description de l'outil d'événement post-build à afficher.
Utilisation dans la génération	Spécifie si cet événement de build est exclu de la génération pour la configuration actuelle.
Fichiers supplémentaires à copier	Spécifie les fichiers supplémentaires à copier sur le système distant. Spécifiez éventuellement des paires de mappage entre l'emplacement local et l'emplacement distant en utilisant une syntaxe telle que la : fulllocalpath1:=fullremotepath1;fulllocalpath2:=fullremotepath2, où un fichier local peut être copié vers l'emplacement distant spécifié sur le système distant.

Événement prébuild distant

Propriété	Description
Ligne de commande	Spécifie une ligne de commande pour l'outil d'événement prébuild à exécuter sur le système distant.
Description	Spécifie une description de l'outil d'événement prébuild à afficher.
Utilisation dans la génération	Spécifie si cet événement de build est exclu de la génération pour la configuration actuelle.
Fichiers supplémentaires à copier	Spécifie les fichiers supplémentaires à copier à partir du système distant. Spécifiez éventuellement des paires de mappage entre l'emplacement distant et l'emplacement local en utilisant une syntaxe telle que la suivante : fullremotepath1:=fulllocalpath1;fullremotepath2:=fulllocalpath2, où un fichier distant peut être copié vers l'emplacement spécifié sur la machine locale.

Événement de préédition des liens distant

Propriété	Description
Ligne de commande	Spécifie une ligne de commande pour l'outil d'événement de préédition des liens à exécuter sur le système distant.
Description	Spécifie une description de l'outil d'événement de préédition des liens à afficher.
Utilisation dans la génération	Spécifie si cet événement de build est exclu de la génération pour la configuration actuelle.

Propriété	Description
Fichiers supplémentaires à copier	Spécifie les fichiers supplémentaires à copier à partir du système distant. Spécifiez éventuellement des paires de mappage entre l'emplacement distant et l'emplacement local en utilisant une syntaxe telle que la suivante : fullremotepath1:=fullllocalpath1;fullremotepath2:=fullllocalpath2, où un fichier distant peut être copié vers l'emplacement spécifié sur la machine locale.

Événement post-build distant

Propriété	Description
Ligne de commande	Spécifie une ligne de commande pour l'outil d'événement post-build à exécuter sur le système distant.
Description	Spécifie une description de l'outil d'événement post-build à afficher.
Utilisation dans la génération	Spécifie si cet événement de build est exclu de la génération pour la configuration actuelle.
Fichiers supplémentaires à copier	Spécifie les fichiers supplémentaires à copier à partir du système distant. Spécifiez éventuellement des paires de mappage entre l'emplacement distant et l'emplacement local en utilisant une syntaxe telle que la suivante : fullremotepath1:=fullllocalpath1;fullremotepath2:=fullllocalpath2, où un fichier distant peut être copié vers l'emplacement spécifié sur la machine locale.

Étape de build personnalisée, propriétés (Linux C++)

Article • 16/06/2023

Propriété	Description
Ligne de commande	Commande à exécuter par l'étape de génération personnalisée.
Description	Message qui s'affiche lors de l'exécution de l'étape de génération personnalisée.
Sorties	Fichier de sortie généré lors de l'étape de génération personnalisée. Ce paramètre est requis afin que les générations incrémentielles fonctionnent correctement.
Dépendances supplémentaires	Liste délimitée par les points-virgules de tous les fichiers d'entrée supplémentaires à utiliser pour l'étape de génération personnalisée.
Exécution après et Exécution avant	Ces options définissent le moment de l'exécution de l'étape de génération personnalisée dans le processus de génération, par rapport aux cibles répertoriées. Les cibles le plus souvent répertoriées sont BuildGenerateSources, BuildCompile, et BuildLink, car elles constituent des étapes majeures dans le processus de génération. Les cibles souvent répertoriées sont Midl, CLCompile, et Link.
Considérer la sortie en tant que contenu	Cette option est significative uniquement pour les applications du Microsoft Store ou Windows Phone, qui comprennent tous les fichiers de contenu du package .appx.

Projet Makefile, propriétés (Linux C++)

Article • 03/04/2023 • 4 minutes de lecture

Il s'agit d'une liste partielle des propriétés disponibles dans un projet Makefile Linux. Beaucoup de propriétés de projet Makefile sont identiques aux propriétés de projet d'application de console C++ Linux.

Général

Propriété	Description	Choices
Répertoire de sortie	Spécifie un chemin relatif vers le répertoire de fichiers de sortie ; peut inclure des variables d'environnement.	
Répertoire intermédiaire	Spécifie un chemin relatif vers le répertoire de fichiers intermédiaire ; peut inclure des variables d'environnement.	
Fichier journal de génération	Spécifie le fichier journal de génération à utiliser quand la journalisation de la génération est activée.	
Type de configuration	Spécifie le type de sortie généré par cette configuration.	**Bibliothèque dynamique (.so)** : bibliothèque dynamique (.so) **Bibliothèque statique (.a)** : bibliothèque statique (.a) **Application (.out)** : application (.out) **Makefile** : makefile
Machine de build distante	Machine ou appareil cible à utiliser pour la génération, le déploiement et le débogage à distance.	
Répertoire racine de build distant	Spécifie le chemin d'un répertoire sur la machine ou l'appareil distant.	
Répertoire de projet de build distant	Spécifie le chemin d'un répertoire sur la machine ou l'appareil distant du projet.	

Débogage

Consultez Débogage, propriétés (Linux C++)

Copier les sources

Consultez Copier les sources, propriétés de projet (Linux C++).

Événements de build

Événement prébuild

Propriété	Description
Ligne de commande	Spécifie une ligne de commande pour l'outil d'événement prébuild à exécuter.
Description	Spécifie une description de l'outil d'événement prébuild à afficher.
Utilisation dans la génération	Spécifie si cet événement de build est exclu de la génération pour la configuration actuelle.
Fichiers supplémentaires à copier	Spécifie les fichiers supplémentaires à copier sur le système distant. Vous pouvez éventuellement fournir la liste sous forme de paires de mappage entre l'emplacement local et l'emplacement distant en utilisant la syntaxe suivante : fulllocalpath1:=fullremotepath1;fulllocalpath2:=fullremotepath2, où un fichier local peut être copié vers l'emplacement distant spécifié sur le système distant.

Événement post-build

Propriété	Description
Ligne de commande	Spécifie une ligne de commande pour l'outil d'événement postbuild à exécuter.
Description	Spécifie une description de l'outil d'événement post-build à afficher.
Utilisation dans la génération	Spécifie si cet événement de build est exclu de la génération pour la configuration actuelle.
Fichiers supplémentaires à copier	Spécifie les fichiers supplémentaires à copier sur le système distant. Vous pouvez éventuellement fournir la liste sous forme de paires de mappage entre l'emplacement local et l'emplacement distant en utilisant la syntaxe suivante : fulllocalpath1:=fullremotepath1;fulllocalpath2:=fullremotepath2, où un fichier local peut être copié vers l'emplacement distant spécifié sur le système distant.

Événement prébuild distant

Propriété	Description
Ligne de commande	Spécifie une ligne de commande pour l'outil d'événement prébuild à exécuter sur le système distant.
Description	Spécifie une description de l'outil d'événement prébuild à afficher.
Utilisation dans la génération	Spécifie si cet événement de build est exclu de la génération pour la configuration actuelle.
Fichiers supplémentaires à copier	Spécifie les fichiers supplémentaires à copier à partir du système distant. Vous pouvez éventuellement fournir la liste sous forme de paires de mappage entre l'emplacement distant et l'emplacement local en utilisant la syntaxe suivante : fullremotepath1:=fullllocalpath1;fullremotepath2:=fullllocalpath2, où un fichier distant peut être copié vers l'emplacement spécifié sur la machine locale.

Événement post-build distant

Propriété	Description
Ligne de commande	Spécifie une ligne de commande pour l'outil d'événement post-build à exécuter sur le système distant.
Description	Spécifie une description de l'outil d'événement post-build à afficher.
Utilisation dans la génération	Spécifie si cet événement de build est exclu de la génération pour la configuration actuelle.
Fichiers supplémentaires à copier	Spécifie les fichiers supplémentaires à copier à partir du système distant. Vous pouvez éventuellement fournir la liste sous forme de paires de mappage entre l'emplacement distant et l'emplacement local en utilisant la syntaxe suivante : fullremotepath1:=fullllocalpath1;fullremotepath2:=fullllocalpath2, où un fichier distant peut être copié vers l'emplacement spécifié sur la machine locale.

C/C++

IntelliSense

Les propriétés IntelliSense peuvent être définies au niveau du projet ou du fichier pour fournir des indications au moteur IntelliSense. Elles n'impactent pas la compilation.

Propriété	Description

Propriété	Description
Chemin de recherche Include	Spécifie le chemin de recherche Include pour résoudre les fichiers Include.
Fichiers Include forcés	Spécifie les fichiers Include forcés.
Définitions de préprocesseur	Spécifie les définitions du préprocesseur utilisées par les fichiers sources.
Annuler la définition de définitions de préprocesseur	Spécifie l'annulation de la définition d'une ou de plusieurs définitions du préprocesseur. (/U[macro])
Options supplémentaires	Spécifie les commutateurs supplémentaires du compilateur utilisés par IntelliSense lors de l'analyse des fichiers C++.

Build

Propriété	Description
Ligne de commande Build	Spécifie la ligne de commande à exécuter pour la commande 'Build'.
Ligne de commande Rebuild All	Spécifie la ligne de commande à exécuter pour la commande 'Rebuild All'.
Ligne de commande Clean	Spécifie la ligne de commande à exécuter pour la commande 'Clean'.

Build distante

Propriété	Description
Ligne de commande Build	Spécifie la ligne de commande à exécuter pour la commande 'Build'. Cette commande est exécutée sur le système distant.
Ligne de commande Rebuild All	Spécifie la ligne de commande à exécuter pour la commande 'Rebuild All'. Cette commande est exécutée sur le système distant.
Ligne de commande Clean	Spécifie la ligne de commande à exécuter pour la commande 'Clean'. Cette commande est exécutée sur le système distant.
Sorties	Spécifie les sorties générées par la build distante sur le système distant.

Archive distante, propriétés (Linux C++)

Article • 16/06/2023

Propriété	Description
Créer un index d'archive	Crée un index d'archive (comme le fait ranlib). Cette option contribue à accélérer l'édition des liens et à réduire la dépendance au sein de sa propre bibliothèque.
Créer une archive fine	Crée une archive fine. Une archive fine n'incorpore pas les objets, mais contient leurs chemins relatifs. Le passage entre les modes Fin et Normal nécessite la suppression de la bibliothèque existante.
Aucun avertissement à la création	N'affiche pas d'avertissement si ou quand la bibliothèque est créée.
Tronquer l'horodatage	Utilise zéro pour les horodatages et les UID/GID.
Supprimer la bannière de démarrage	N'affiche pas le numéro de version.
Commentaires	Commentaires
Options supplémentaires	Options supplémentaires.
Fichier de sortie	L'option /OUT substitue le nom et l'emplacement par défaut du programme créé par lib.
Programme d'archivage	Spécifie le programme à appeler durant l'édition des liens d'objets statiques, ou le chemin du programme d'archivage sur le système distant.
Délai d'attente du programme d'archivage	Délai d'attente du programme d'archivage distant, en millisecondes.
Copier la sortie	Indique s'il faut copier le fichier de sortie de build du système distant sur la machine locale.

Créer un projet Linux CMake dans Visual Studio

Article • 16/06/2023

Nous vous recommandons d'utiliser CMake pour les projets qui sont multiplateformes ou qui seront rendus open source. Vous pouvez utiliser des projets CMake pour générer et déboguer le même code source sur Windows, le sous-système Windows pour Linux (Windows Subsystem for Linux, WSL) et des systèmes distants.

Avant de commencer

Tout d'abord, vérifiez que la charge de travail Visual Studio Linux est installée, y compris le composant CMake. Il s'agit de la charge de travail de **développement Linux avec C++** dans le programme d'installation de Visual Studio. Si vous n'êtes pas certain qu'elle est installée, consultez Installer la charge de travail Linux C++ dans Visual Studio.

Assurez-vous également que les éléments suivants sont installés sur la machine distante :

- gcc
- gdb
- rsync
- zip
- ninja-build (Visual Studio 2019 ou version ultérieure)

Vous pouvez utiliser Visual Studio 2019 pour générer et déboguer sur un système Linux distant ou WSL. CMake sera appelé sur ce système. Cmake version 3.14 ou ultérieure doit être installé sur la machine cible.

Vérifiez que la machine cible dispose d'une version récente de CMake. Souvent, la version proposée par le gestionnaire de package par défaut d'une distribution n'est pas assez récente pour prendre en charge toutes les fonctionnalités que Visual Studio requiert. Visual Studio 2019 détecte si une version récente de CMake est installée sur le système Linux. Si aucune n'est trouvée, Visual Studio affiche une barre d'informations en haut du volet de l'éditeur. Il propose d'installer CMake pour vous à partir de https://github.com/Microsoft/CMake/releases .

Avec Visual Studio 2019, vous pouvez créer un projet CMake à partir de rien, ou ouvrir un projet CMake existant. Pour créer un projet CMake, suivez les instructions ci-dessous.

Ou passez directement à Ouvrir un dossier de projet CMake si vous avez déjà un projet CMake.

Créer un projet Linux CMake

Pour créer un nouveau projet Linux CMake dans Visual Studio 2019 :

1. Sélectionnez **Fichier** > **Nouveau projet** dans Visual Studio, ou appuyez sur **Ctrl+Maj+N**.
2. Définissez le **Langage** sur **C++** et recherchez « CMake ». Ensuite, choisissez **Suivant**. Entrez un **Nom** et un **Emplacement**, puis choisissez **Créer**.

Vous pouvez également ouvrir votre propre projet CMake dans Visual Studio 2019. La section suivante explique comment procéder.

Visual Studio crée un fichier *CMakeLists.txt* minimal avec uniquement le nom de l'exécutable et la version de CMake minimale requise. Vous pouvez modifier manuellement ce fichier à votre convenance ; Visual Studio ne remplacera jamais vos changements.

Pour vous aider à comprendre, modifier et créer vos scripts CMake dans Visual Studio 2019, consultez les ressources suivantes :

- Documentation dans l'éditeur pour CMake dans Visual Studio
- Navigation dans le code pour les scripts CMake
- Ajouter, supprimer et renommer facilement des fichiers et des cibles dans des projets CMake

Ouvrir un dossier de projet CMake

Lorsque vous ouvrez un dossier contenant un projet CMake existant, Visual Studio utilise des variables dans le cache CMake pour configurer automatiquement IntelliSense et les builds. Les paramètres de configuration et de débogage locaux sont stockés dans des fichiers JSON. Vous pouvez éventuellement partager ces fichiers avec d'autres utilisateurs de Visual Studio.

Visual Studio ne modifie pas les fichiers *CMakeLists.txt*. Cela permet aux autres personnes travaillant sur le même projet de continuer à utiliser leurs outils existants. Visual Studio régénère le cache lorsque vous enregistrez des modifications apportées au fichier *CMakeLists.txt* ou, dans certains cas, au fichier *CMakeSettings.json*. Si vous utilisez une configuration de **Cache existant**, Visual Studio ne modifie pas le cache.

Pour obtenir des informations générales sur la prise en charge de CMake dans Visual Studio, consultez Projets CMake dans Visual Studio. Lisez cet article avant de continuer ici.

Pour commencer, choisissez **Fichier**>**Ouvrir**>**Dossier** dans le menu principal, ou tapez `devenv.exe <foldername>` dans une fenêtre d'invite de commandes développeur. Le dossier que vous ouvrez doit contenir un fichier *CMakeLists.txt*, ainsi que votre code source.

L'exemple suivant montre un fichier *CMakeLists.txt* et un fichier .cpp simples :

C++

```
// hello.cpp

#include <iostream>

int main(int argc, char* argv[])
{
    std::cout << "Hello from Linux CMake \n";
}
```

CMakeLists.txt :

txt

```
cmake_minimum_required(VERSION 3.8)
project (hello-cmake)
add_executable(hello-cmake hello.cpp)
```

Étapes suivantes

Configurer un projet CMake Linux

Voir aussi

Projets CMake dans Visual Studio

Configurer un projet CMake Linux dans Visual Studio

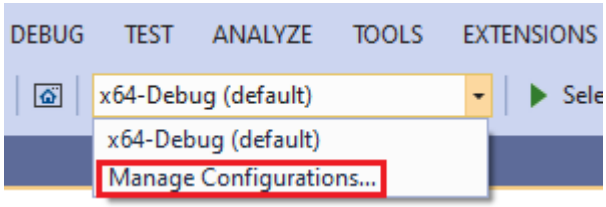
Article • 16/06/2023

Cette rubrique explique comment ajouter une configuration Linux à un projet CMake qui cible un système Linux distant ou un Sous-système Windows pour Linux (WSL). Elle fait suite à la série qui commençait par Créer un projet CMake Linux dans Visual Studio. Si vous utilisez plutôt MSBuild, consultez Configurer un projet MSBuild Linux dans Visual Studio.

Ajouter une configuration Linux

Une configuration peut être utilisée pour cibler différentes plateformes (Windows, WSL, système distant) avec le même code source. Une configuration est également utilisée pour définir vos compilateurs, passer des variables d'environnement et personnaliser la façon dont CMake est appelé. Le *fichier CMakeSettings.json* spécifie certaines ou toutes les propriétés répertoriées dans Personnaliser les paramètres CMake, ainsi que d'autres propriétés qui contrôlent les paramètres de build sur la machine Linux distante.

Pour changer les paramètres CMake par défaut dans Visual Studio 2019 ou version ultérieure, dans la barre d'outils principale, ouvrez la liste déroulante **Configuration**, puis choisissez **Gérer les configurations**.



Cette commande ouvre l'**éditeur de paramètres CMake** qui vous permet de modifier le fichier *CMakeSettings.json* situé dans le dossier de projet racine. Vous pouvez également ouvrir le fichier avec l'éditeur JSON en cliquant sur le bouton **Modifier json** en haut à droite de la boîte de dialogue **Paramètres CMake**. Pour plus d'informations, consultez Personnaliser les paramètres CMake.

La configuration par défaut de Linux-Debug dans Visual Studio 2019 versions 16.1 et ultérieures ressemble à ceci :

JSON

```
{
  "configurations": [
```

```
    {
      "name": "Linux-GCC-Debug",
      "generator": "Ninja",
      "configurationType": "Debug",
      "cmakeExecutable": "cmake",
      "remoteCopySourcesExclusionList": [ ".vs", ".git", "out" ],
      "cmakeCommandArgs": "",
      "buildCommandArgs": "",
      "ctestCommandArgs": "",
      "inheritEnvironments": [ "linux_x64" ],
      "remoteMachineName": "${defaultRemoteMachineName}",
      "remoteCMakeListsRoot":
"$HOME/.vs/${projectDirName}/${workspaceHash}/src",
      "remoteBuildRoot":
"$HOME/.vs/${projectDirName}/${workspaceHash}/out/build/${name}",
      "remoteInstallRoot":
"$HOME/.vs/${projectDirName}/${workspaceHash}/out/install/${name}",
      "remoteCopySources": true,
      "rsyncCommandArgs": "-t --delete --delete-excluded",
      "remoteCopyBuildOutput": false,
      "remoteCopySourcesMethod": "rsync",
      "variables": []
    }
  ]
}
```

Dans Visual Studio 2019 version 16.6 ou ultérieure, Ninja est le générateur par défaut pour les configurations ciblant un système distant ou WSL, par opposition aux Makefiles Unix. Pour plus d'informations, consultez ce billet sur le Blog de l'équipe C++.

Pour plus d'informations sur ces paramètres, consultez la Référence de CMakeSettings.json.

Quand vous effectuez une build :

- Si vous ciblez un système distant, Visual Studio choisit le premier système distant dans la liste sous **Outils**>**Options**>**Multiplateforme**>**Gestionnaire de connexions** par défaut pour les cibles distantes.
- Si aucune connexion distante n'est trouvée, vous êtes invité à en créer une. Pour plus d'informations, consultez Se connecter à un ordinateur Linux distant.

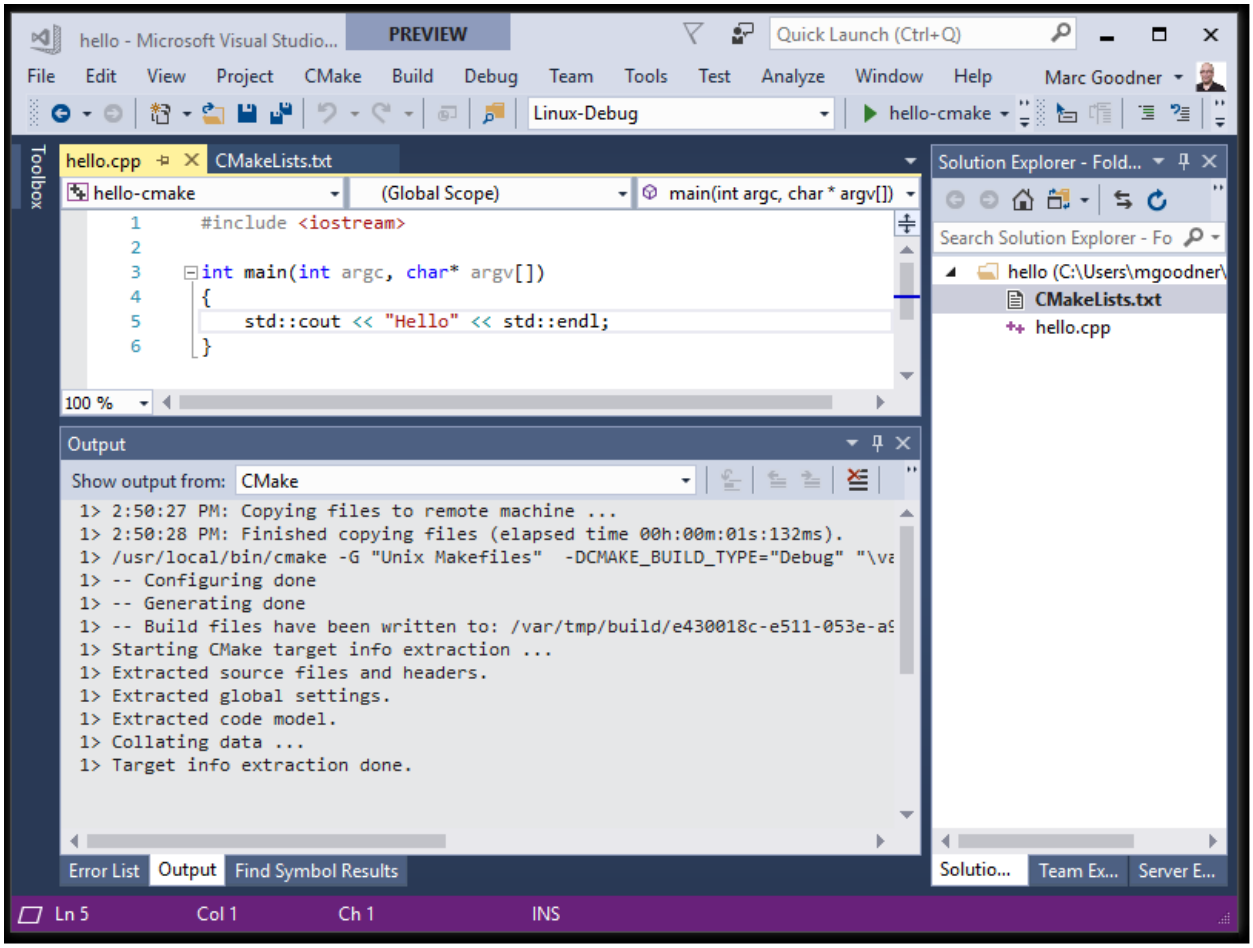
Choisir une cible Linux

Lorsque vous ouvrez un dossier de projet CMake, Visual Studio analyse le fichier **CMakeLists.txt** et spécifie la cible Windows de *x86-Debug*. Pour cibler un système Linux distant, vous allez modifier les paramètres du projet en fonction de votre compilateur

Linux. Par exemple, si vous utilisez GCC sur Linux et compilez avec des informations de débogage, choisissez : **Linux-GCC-Debug** ou **Linux-GCC-Release**.

Si vous spécifiez une cible Linux distante, votre source est copiée sur le système distant.

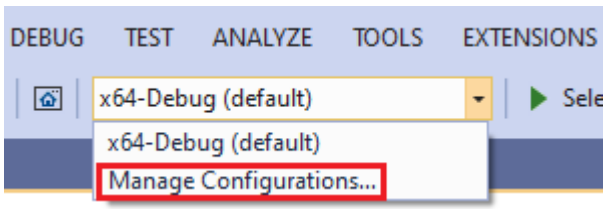
Une fois que vous sélectionnez une cible, CMake s'exécute automatiquement sur le système Linux afin de générer le cache CMake pour votre projet :



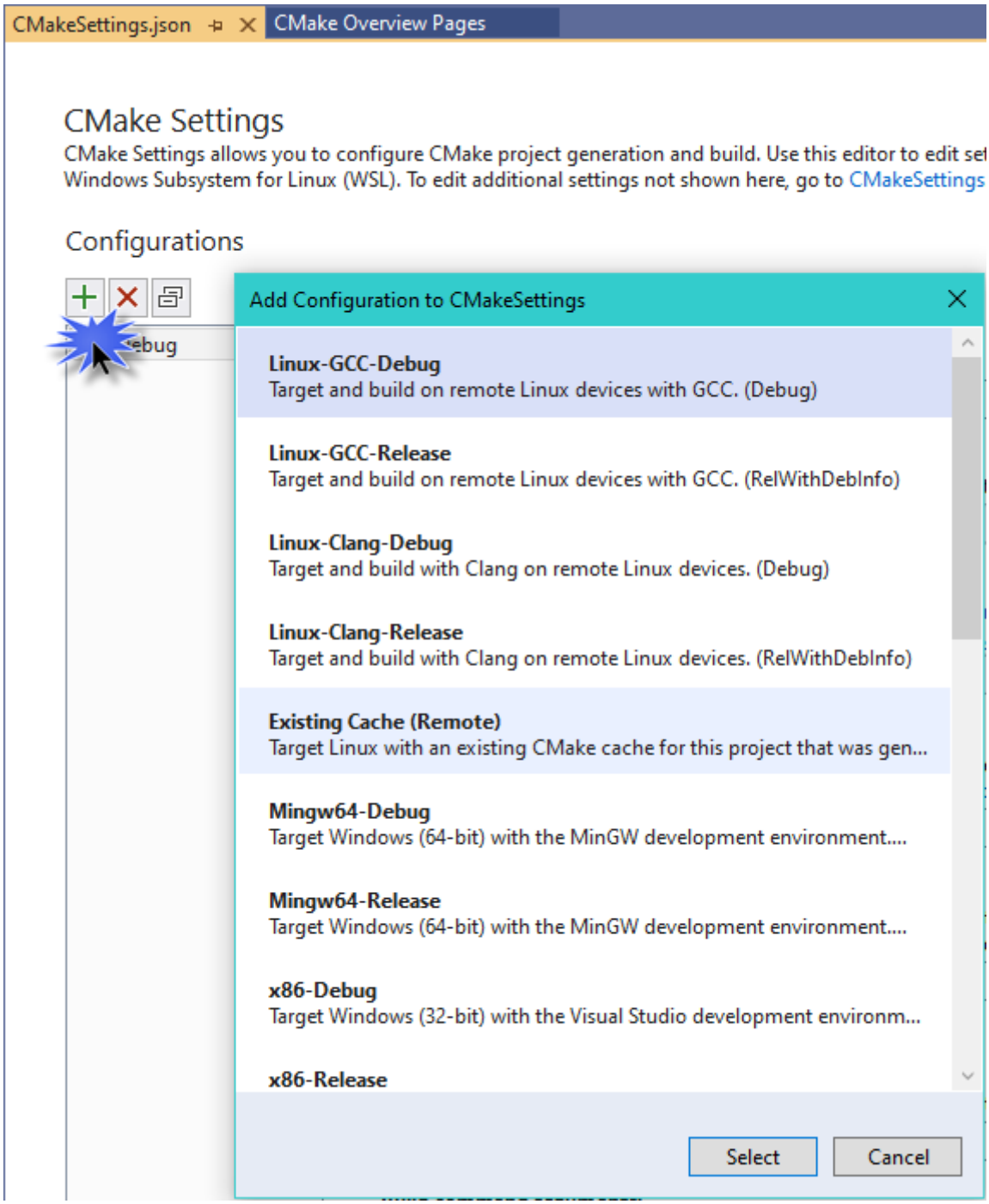
Sous-système Windows pour Linux

Si vous ciblez le Sous-système Windows pour Linux (WSL), vous n'avez pas besoin d'ajouter une connexion à distance.

Pour cibler WSL, dans la liste déroulante de configuration de la barre d'outils principale, sélectionnez **Gérer les configurations** :



La fenêtre **CMakeSettings.json** s'affiche.



Appuyez sur **Ajouter une configuration** (le bouton vert « + »), puis choisissez **Linux-GCC-Debug** ou **Linux-GCC-Release** si vous utilisez GCC. Utilisez les variantes Clang si vous utilisez l'ensemble d'outils Clang/LLVM. Appuyez sur **Sélectionner, puis sur Ctrl+S** pour enregistrer la configuration.

Visual Studio 2019 version 16.1 Lorsque vous ciblez WSL, Visual Studio n'a pas besoin de copier les fichiers sources et de gérer deux copies synchrones de votre arborescence de build, car le compilateur sur Linux a un accès direct à vos fichiers sources dans le système de fichiers Windows monté.

IntelliSense

IntelliSense pour C++ nécessite l'accès aux en-têtes C++ référencés par vos fichiers sources C++. Visual Studio utilise automatiquement les en-têtes référencés par un projet CMake de Linux vers Windows pour fournir une expérience IntelliSense d'une fidélité optimale. Pour plus d'informations, consultez IntelliSense pour les en-têtes distants.

Définition des paramètres régionaux

Les paramètres de langue Visual Studio ne sont pas propagés aux cibles Linux, car Visual Studio ne gère ou configure pas les packages installés. Les messages affichés dans la fenêtre Sortie, tels que les erreurs de build, sont présentés en utilisant la langue et les paramètres régionaux de la cible Linux. Vous devez configurer vos cibles Linux pour les paramètres régionaux souhaités.

Paramètres supplémentaires

Utilisez les paramètres suivants pour exécuter des commandes sur le système Linux avant et après la build, et avant la génération CMake. Les valeurs peuvent correspondre à n'importe quelle commande valide sur le système distant. La sortie est redirigée vers Visual Studio.

JSON

```
{
      "remotePrebuildCommand": "",
      "remotePreGenerateCommand": "",
      "remotePostbuildCommand": "",
}
```

Étapes suivantes

Configurer des sessions de débogage CMake

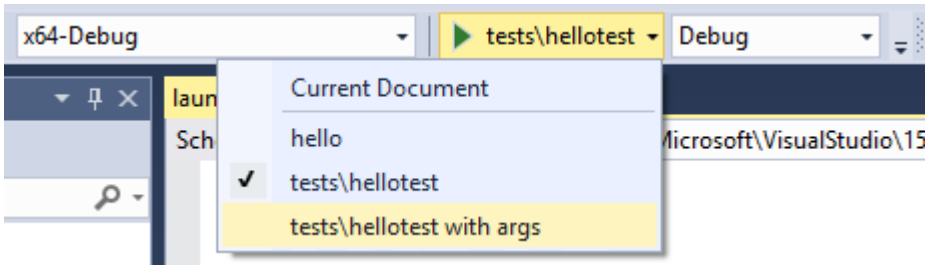
Voir aussi

Utilisation de propriétés de projet
Personnaliser les paramètres CMake
Informations de référence sur la configuration prédéfinie de CMake

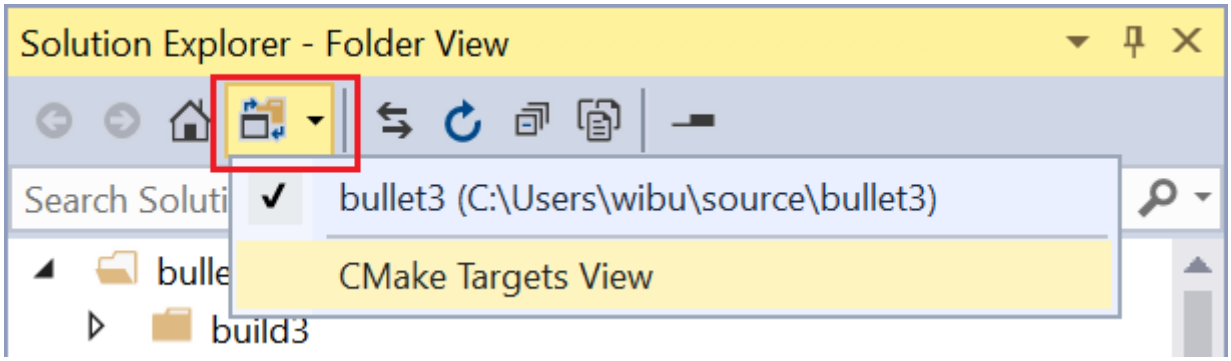
Configurer des sessions de débogage CMake

Article • 16/06/2023

Toutes les cibles CMake exécutables sont affichées dans la liste déroulante **Élément de démarrage** de la barre d'outils. Sélectionnez-en une pour démarrer une session de débogage et lancer le débogueur.



Vous pouvez également démarrer une session de débogage à partir de Explorateur de solutions. Tout d'abord, basculez vers la **vue Cibles CMake** dans la fenêtre **Explorateur de solutions**.

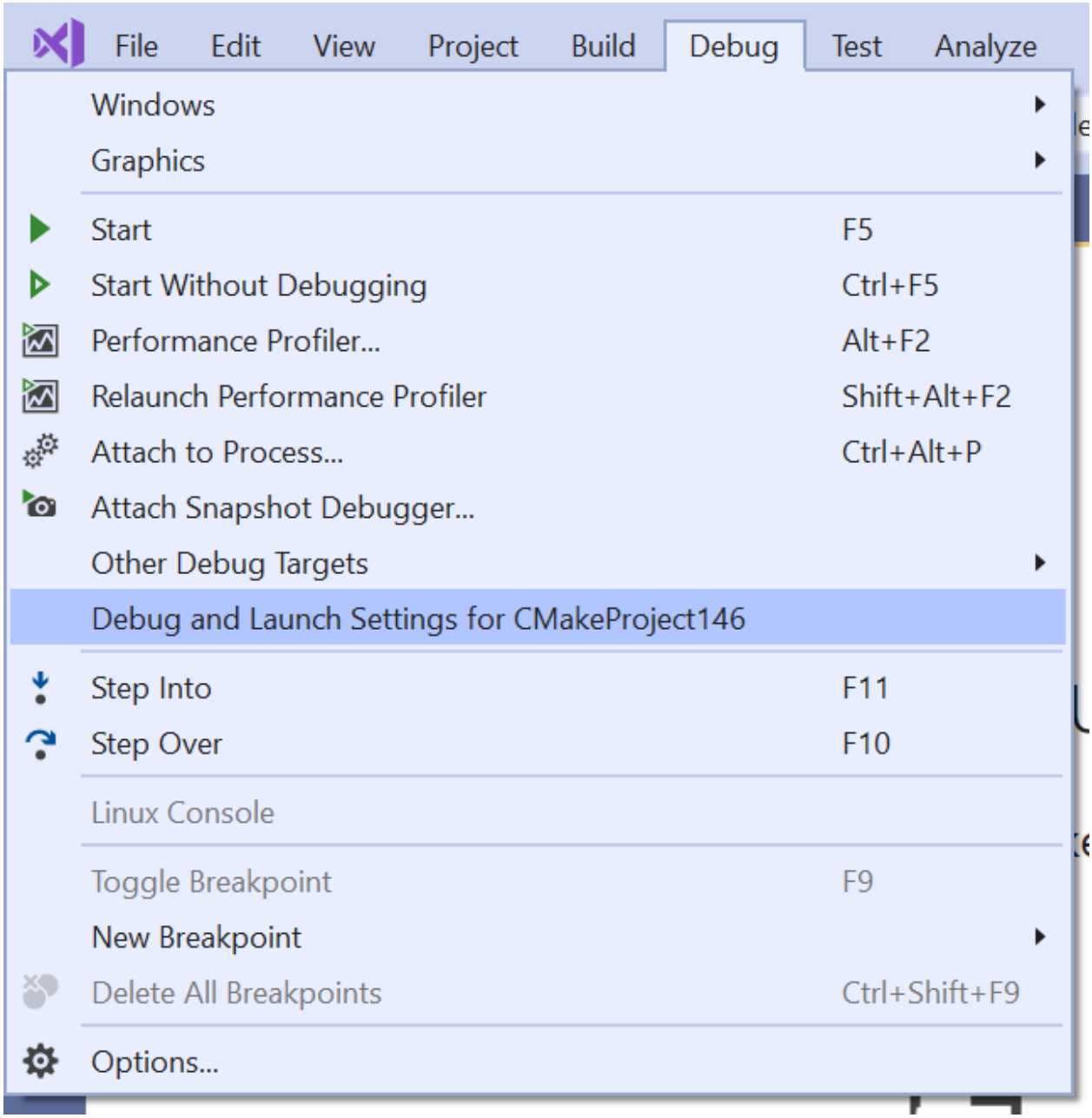


Cliquez ensuite avec le bouton droit sur un exécutable, puis sélectionnez **Déboguer**. Cette commande démarre automatiquement le débogage de la cible sélectionnée en fonction de votre configuration active.

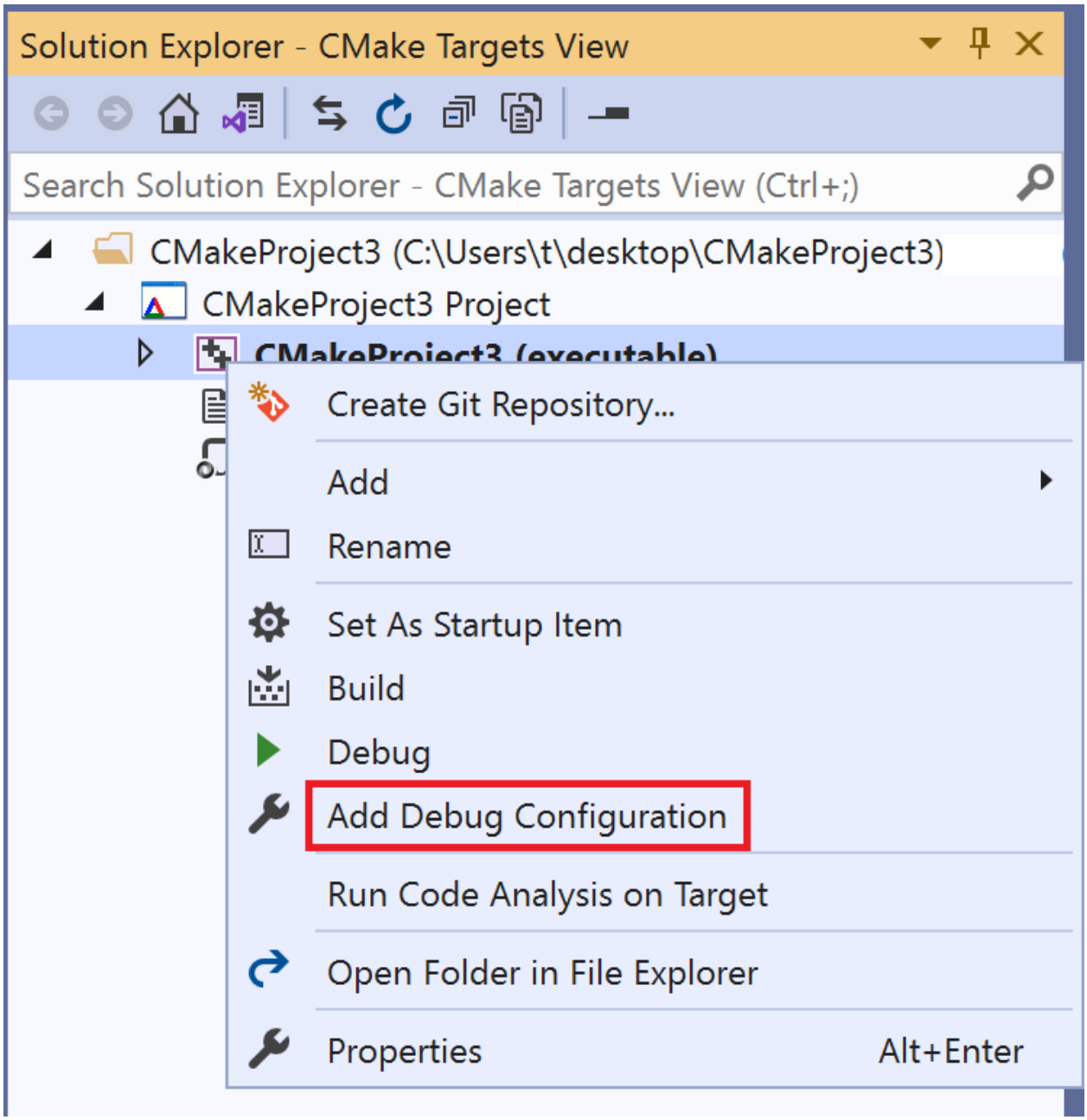
Personnaliser les paramètres du débogueur

Vous pouvez personnaliser les paramètres du débogueur pour n'importe quelle cible CMake exécutable dans votre projet. Ils se trouvent dans un fichier de configuration appelé *launch.vs.json*, situé dans un `.vs` dossier à la racine de votre projet. Un fichier de configuration de lancement est utile dans la plupart des scénarios de débogage, car vous pouvez configurer et enregistrer vos détails de configuration de débogage. Ce fichier comporte trois points d'entrée :

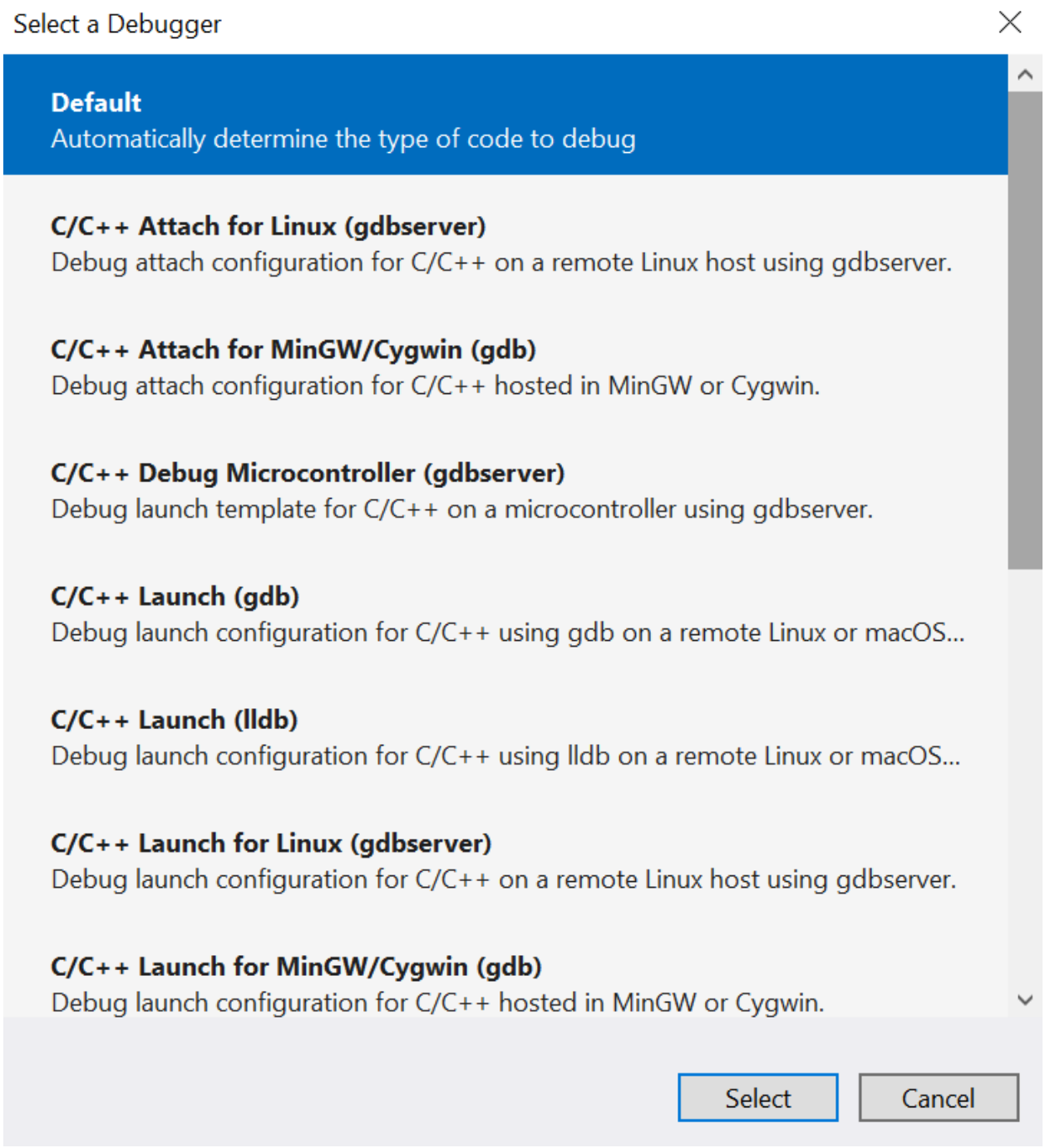
- **Menu Débogage :** Sélectionnez Paramètres de débogage **et de lancement pour ${activeDebugTarget} dans le menu main pour personnaliser la configuration de débogage spécifique à votre cible de débogage active.>** Si aucune cible de débogage n'est sélectionnée, cette option est grisée.



- **Vue Cibles :** Accédez à **La vue Cibles** dans Explorateur de solutions. Cliquez ensuite avec le bouton droit sur une cible de débogage et sélectionnez **Ajouter une configuration de débogage** pour personnaliser la configuration de débogage spécifique à la cible sélectionnée.



- **CMakeLists.txt racine :** Cliquez avec le bouton droit sur un *CMakeLists.txt* racine et sélectionnez **Ajouter une configuration de débogage** pour ouvrir la boîte de dialogue **Sélectionner un débogueur** . La boîte de dialogue vous permet d'ajouter *n'importe quel* type de configuration de débogage, mais vous devez spécifier manuellement la cible CMake à appeler via la `projectTarget` propriété .



Vous pouvez modifier le fichier *launch.vs.json* pour créer des configurations de débogage pour un nombre quelconque de cibles CMake. Lorsque vous enregistrez le fichier, Visual Studio crée une entrée pour chaque nouvelle configuration dans la liste déroulante **Élément de démarrage** .

Clés de référence dans CMakeSettings.json

Pour référencer une clé dans un fichier *CMakeSettings.json* , `cmake.` ajoutez-y dans *launch.vs.json*. L'exemple suivant montre un fichier *launch.vs.json* simple qui extrait la valeur de la `remoteCopySources` clé dans le fichier *CMakeSettings.json* pour la configuration actuellement sélectionnée :

JSON

```
{
  "version": "0.2.1",
  "configurations": [
    {
      "type": "default",
      "project": "CMakeLists.txt",
      "projectTarget": "CMakeHelloWorld.exe (Debug\\CMakeHelloWorld.exe)",
      "name": "CMakeHelloWorld.exe (Debug\\CMakeHelloWorld.exe)",
      "args": ["${cmake.remoteCopySources}"]
    }
  ]
}
```

Les variables d'environnement définies dans *CMakeSettings.json* peuvent également être utilisées dans launch.vs.json à l'aide de la syntaxe `${env.VARIABLE_NAME}`. Dans Visual Studio 2019 version 16.4 et ultérieures, les cibles de débogage sont automatiquement lancées à l'aide de l'environnement que vous spécifiez dans *CMakeSettings.json*. Vous pouvez annuler la définition d'une variable d'environnement en lui affectant la **valeur Null**.

Référence Launch.vs.json

Il existe de nombreuses propriétés *launch.vs.json* pour prendre en charge tous vos scénarios de débogage. Les propriétés suivantes sont communes à toutes les configurations de débogage, à la fois distantes et locales :

- `projectTarget`: spécifie la cible CMake à appeler lors de la génération du projet. Visual Studio remplit automatiquement cette propriété si vous entrez *launch.vs.json* à partir du **menu Débogage** ou **de la vue Cibles**. Cette valeur doit correspondre au nom d'une cible de débogage existante répertoriée dans la liste déroulante **Élément de démarrage** .

- `env`: variables d'environnement supplémentaires à ajouter à l'aide de la syntaxe :

  JSON

  ```
  "env": {
        "DEBUG_LOGGING_LEVEL": "trace;info",
        "ENABLE_TRACING": "true"
      }
  ```

- `args`: arguments de ligne de commande passés au programme à déboguer.

Référence Launch.vs.json pour les projets distants et WSL

Dans Visual Studio 2019 version 16.6, nous avons ajouté une nouvelle configuration de débogage de `type: cppgdb` pour simplifier le débogage sur les systèmes distants et WSL. Les anciennes configurations de débogage de `type: cppdbg` sont toujours prises en charge.

Type de configuration `cppgdb`

- `name`: nom convivial permettant d'identifier la configuration dans la liste déroulante **Élément de démarrage** .
- `project`: spécifie le chemin d'accès relatif au fichier projet. Normalement, vous n'avez pas besoin de modifier ce chemin lors du débogage d'un projet CMake.
- `projectTarget`: spécifie la cible CMake à appeler lors de la génération du projet. Visual Studio remplit automatiquement cette propriété si vous entrez *launch.vs.json* à partir du **menu Débogage** ou **de la vue Cibles**. Cette valeur cible doit correspondre au nom d'une cible de débogage existante répertoriée dans la liste déroulante **Élément de démarrage** .
- `debuggerConfiguration`: indique l'ensemble de valeurs par défaut de débogage à utiliser. Dans Visual Studio 2019 version 16.6, la seule option valide est `gdb`. Visual Studio 2019 version 16.7 ou ultérieure prend également en charge `gdbserver`.
- `args`: arguments de ligne de commande transmis au démarrage au programme en cours de débogage.
- `env`: variables d'environnement supplémentaires passées au programme en cours de débogage. Par exemple : `{"DISPLAY": "0.0"}`.
- `processID`: ID de processus Linux auquel attacher. Utilisé uniquement lors de l'attachement à un processus distant. Pour plus d'informations, consultez Résoudre les problèmes d'attachement à des processus à l'aide de GDB .

Options supplémentaires pour la `gdb` configuration

- `program` : a la valeur par défaut `"${debugInfo.fullTargetPath}"`. Chemin Unix de l'application à déboguer. Obligatoire uniquement s'il est différent de l'exécutable cible dans l'emplacement de génération ou de déploiement.
- `remoteMachineName` : a la valeur par défaut `"${debugInfo.remoteMachineName}"`. Nom du système distant qui héberge le programme à déboguer. Obligatoire uniquement s'il est différent du système de build. Doit avoir une entrée existante

dans le Gestionnaire des connexions. **Appuyez sur Ctrl+Espace** pour afficher la liste de toutes les connexions distantes existantes.

- `cwd` : a la valeur par défaut `"${debugInfo.defaultWorkingDirectory}"`. Chemin Unix du répertoire sur le système distant où `program` est exécuté. Le répertoire doit exister.
- `gdbpath` : a la valeur par défaut `/usr/bin/gdb`. Chemin d'accès Unix complet au `gdb` utilisé pour le débogage. Obligatoire uniquement si vous utilisez une version personnalisée de `gdb`.
- `preDebugCommand`: commande Linux à exécuter immédiatement avant d'appeler `gdb`. `gdb` ne démarre pas tant que la commande n'est pas terminée. Vous pouvez utiliser l'option pour exécuter un script avant l'exécution de `gdb`.

Options supplémentaires autorisées avec la `gdbserver` configuration (16.7 ou version ultérieure)

- `program` : a la valeur par défaut `"${debugInfo.fullTargetPath}"`. Chemin Unix de l'application à déboguer. Obligatoire uniquement s'il est différent de l'exécutable cible dans l'emplacement de génération ou de déploiement.

  > **Conseil**
  >
  > Le déploiement n'est pas encore pris en charge pour les scénarios de compilation croisée locale. Si vous effectuez une compilation croisée sur Windows (par exemple, en utilisant un compilateur croisé sur Windows pour créer un exécutable ARM Linux), vous devez copier manuellement le fichier binaire à l'emplacement spécifié par `program` sur l'ordinateur ARM distant avant le débogage.

- `remoteMachineName` : a la valeur par défaut `"${debugInfo.remoteMachineName}"`. Nom du système distant qui héberge le programme à déboguer. Obligatoire uniquement s'il est différent du système de build. Doit avoir une entrée existante dans le Gestionnaire des connexions. **Appuyez sur Ctrl+Espace** pour afficher la liste de toutes les connexions distantes existantes.

- `cwd` : a la valeur par défaut `"${debugInfo.defaultWorkingDirectory}"`. Chemin d'accès Unix complet au répertoire sur le système distant où `program` est exécuté. Le répertoire doit exister.

- `gdbPath` : a la valeur par défaut `${debugInfo.vsInstalledGdb}`. Chemin d'accès Windows complet au `gdb` utilisé pour déboguer. La valeur par défaut est installée

`gdb` avec la charge de travail développement Linux avec C/C++.

- `gdbserverPath` : a la valeur par défaut `usr/bin/gdbserver`. Chemin Unix complet du `gdbserver` utilisé pour déboguer.

- `preDebugCommand`: commande Linux à exécuter immédiatement avant de démarrer `gdbserver`. `gdbserver` ne démarre pas tant que la commande n'est pas terminée.

Options de déploiement

Utilisez les options suivantes pour séparer votre machine de build (définie dans CMakeSettings.json) de votre machine de débogage distante.

- `remoteMachineName`: machine de débogage distante. Obligatoire uniquement s'il est différent de la machine de build. Doit avoir une entrée existante dans le Gestionnaire des connexions. **Appuyez sur Ctrl+Espace** pour afficher la liste de toutes les connexions distantes existantes.
- `disableDeploy` : a la valeur par défaut **`false`**. Indique si la séparation build/débogage est désactivée. Quand , **`false`**cette option permet à la génération et au débogage de se produire sur deux ordinateurs distincts.
- `deployDirectory`: chemin Unix complet vers le répertoire dans `remoteMachineName` lequel l'exécutable est copié.
- `deploy`: tableau de paramètres de déploiement avancés. Vous devez uniquement configurer ces paramètres lorsque vous souhaitez un contrôle plus granulaire sur le processus de déploiement. Par défaut, seuls les fichiers nécessaires au processus de débogage sont déployés sur l'ordinateur de débogage distant.
  - `sourceMachine`: machine à partir de laquelle le fichier ou le répertoire est copié. **Appuyez sur Ctrl+Espace** pour afficher la liste de toutes les connexions distantes stockées dans le Gestionnaire des connexions. Lors de la génération en mode natif sur WSL, cette option est ignorée.
  - `targetMachine`: machine sur laquelle le fichier ou le répertoire est copié. **Appuyez sur Ctrl+Espace** pour afficher la liste de toutes les connexions distantes stockées dans le Gestionnaire des connexions.
  - `sourcePath`: emplacement du fichier ou du répertoire sur `sourceMachine`.
  - `targetPath`: emplacement du fichier ou du répertoire sur `targetMachine`.
  - `deploymentType`: description du type de déploiement. `LocalRemote` et `RemoteRemote` sont pris en charge. `LocalRemote` signifie la copie du système de fichiers local vers le système distant spécifié par `remoteMachineName` dans *launch.vs.json*. `RemoteRemote` signifie la copie à partir du système de build distant

spécifié dans *CMakeSettings.json* vers le système distant différent spécifié dans *launch.vs.json*.
    - `executable`: indique si le fichier déployé est un exécutable.

Exécuter des commandes personnalisées `gdb`

Visual Studio prend en charge l'exécution de commandes personnalisées `gdb` pour interagir directement avec le débogueur sous-jacent. Pour plus d'informations, consultez Exécution de commandes lldb personnaliséesgdb.

Activation de la journalisation

Activez la journalisation MIEngine pour voir quelles commandes sont envoyées à `gdb`, quelle sortie `gdb` retourne et combien de temps prend chaque commande. En savoir plus

Type de configuration `cppdbg`

Les options suivantes peuvent être utilisées lors du débogage sur un système distant ou WSL à l'aide du type de `cppdbg` configuration. Dans Visual Studio 2019 version 16.6 ou ultérieure, le type `cppgdb` de configuration est recommandé.

- `name`: nom convivial permettant d'identifier la configuration dans la liste déroulante **Élément de démarrage** .
- `project`: spécifie le chemin relatif du fichier projet. Normalement, vous n'avez pas besoin de modifier cette valeur lors du débogage d'un projet CMake.
- `projectTarget`: spécifie la cible CMake à appeler lors de la génération du projet. Visual Studio remplit automatiquement cette propriété si vous entrez *launch.vs.json* à partir du **menu Déboguer** ou de la **vue Cibles**. Cette valeur doit correspondre au nom d'une cible de débogage existante répertoriée dans la liste déroulante **Élément de démarrage** .
- `args`: arguments de ligne de commande transmis au démarrage au programme en cours de débogage.
- `processID`: ID de processus Linux à attacher. Utilisé uniquement lors de l'attachement à un processus distant. Pour plus d'informations, consultez Résoudre les problèmes d'attachement à des processus à l'aide de GDB .

- `program` : a la valeur par défaut `"${debugInfo.fullTargetPath}"`. Chemin Unix de l'application à déboguer. Obligatoire uniquement s'il est différent de l'exécutable cible dans l'emplacement de génération ou de déploiement.
- `remoteMachineName` : a la valeur par défaut `"${debugInfo.remoteMachineName}"`. Nom du système distant qui héberge le programme à déboguer. Obligatoire uniquement s'il est différent du système de build. Doit avoir une entrée existante dans le Gestionnaire des connexions. **Appuyez sur Ctrl+Espace** pour afficher la liste de toutes les connexions distantes existantes.
- `cwd` : a la valeur par défaut `"${debugInfo.defaultWorkingDirectory}"`. Chemin d'accès Unix complet au répertoire sur le système distant où `program` est exécuté. Le répertoire doit exister.
- `environment`: variables d'environnement supplémentaires passées au programme en cours de débogage. Par exemple,

  JSON

  ```
  "environment": [
      {
        "name": "ENV1",
        "value": "envvalue1"
      },
      {
        "name": "ENV2",
        "value": "envvalue2"
      }
    ]
  ```

- `pipeArgs`: tableau d'arguments de ligne de commande transmis au programme de canal pour configurer la connexion. Le programme de canal est utilisé pour relayer l'entrée/sortie standard entre Visual Studio et `gdb`. La plupart de ce tableau **n'a pas besoin d'être personnalisé** lors du débogage de projets CMake. L'exception est , `${debuggerCommand}`qui se lance `gdb` sur le système distant. Il peut être modifié pour :
  - Exportez la valeur de la variable d'environnement DISPLAY sur votre système Linux. Dans l'exemple suivant, cette valeur est `:1`.

    JSON

    ```
    "pipeArgs": [
        "/s",
        "${debugInfo.remoteMachineId}",
    ```

```
    "/p",
    "${debugInfo.parentProcessId}",
    "/c",
    "export DISPLAY=:1;${debuggerCommand}",
    "--tty=${debugInfo.tty}"
  ],
```

- Exécutez un script avant l'exécution de `gdb`. Vérifiez que les autorisations d'exécution sont définies sur votre script.

```
JSON

"pipeArgs": [
    "/s",
    "${debugInfo.remoteMachineId}",
    "/p",
    "${debugInfo.parentProcessId}",
    "/c",
    "/path/to/script.sh;${debuggerCommand}",
    "--tty=${debugInfo.tty}"
  ],
```

- `stopOnEntry`: booléen qui spécifie s'il faut interrompre le processus dès le lancement du processus. La valeur par défaut est false.
- `visualizerFile`: fichier .natvis à utiliser lors du débogage de ce processus. Cette option n'est pas compatible avec `gdb` l'impression. Également défini `showDisplayString` lorsque vous définissez cette propriété.
- `showDisplayString`: booléen qui active la chaîne d'affichage lorsqu'un `visualizerFile` est spécifié. La définition de **`true`** cette option sur peut ralentir les performances pendant le débogage.
- `setupCommands`: une ou plusieurs `gdb` commandes à exécuter pour configurer le débogueur sous-jacent.
- `miDebuggerPath`: chemin d'accès complet à `gdb`. Lorsqu'il n'est pas spécifié, Visual Studio recherche d'abord PATH pour le débogueur.
- Enfin, toutes les options de déploiement définies pour le `cppgdb` type de configuration peuvent également être utilisées par le `cppdbg` type de configuration.

Déboguer à l'aide de `gdbserver`

Vous pouvez configurer la `cppdbg` configuration pour déboguer à l'aide de `gdbserver`. Vous trouverez plus d'informations et un exemple de configuration de lancement dans le billet de blog de l'équipe Microsoft C++ Débogage de projets Linux CMake avec gdbserver.

Voir aussi

Projets CMake dans Visual Studio
Configurer un projet CMake Linux
Se connecter à un ordinateur Linux distant
Personnaliser des paramètres de génération CMake
Configurer des sessions de débogage CMake
Déployer, exécuter et déboguer un projet Linux
Informations de référence sur la configuration prédéfinie de CMake

Tutoriel : Créer des projets multiplateformes C++ dans Visual Studio

Article • 03/04/2023 • 11 minutes de lecture

Le développement en C et C++ dans Visual Studio ne cible plus seulement les environnements Windows. Ce tutoriel montre comment utiliser Visual Studio pour le développement multiplateforme C++ sur Windows et Linux. Basé sur CMake, vous n'avez donc pas besoin de créer ou de générer des projets Visual Studio. Lorsque vous ouvrez un dossier qui contient un fichier CMakeLists.txt, Visual Studio configure automatiquement les paramètres IntelliSense et de génération. Vous pouvez rapidement commencer à modifier, générer et déboguer votre code localement sur Windows. Ensuite, basculez votre configuration pour faire de même sur Linux, le tout à partir de Visual Studio.

Dans ce tutoriel, vous allez apprendre à :

- ✓ Cloner un projet CMake open source à partir de GitHub
- ✓ Ouvrir le projet dans Visual Studio
- ✓ Générer et déboguer un exécutable cible sur Windows
- ✓ Ajouter une connexion à une machine Linux
- ✓ Générer et déboguer la même cible sur Linux

Prérequis

- Configurer Visual Studio pour le développement multiplateforme en C++
  - Tout d'abord, installez Visual Studio et choisissez les charges de **travail Développement de bureau avec C++** et **Développement Linux avec C++**. Cette installation minimale n'est que de 3 Go. En fonction de la vitesse de téléchargement, l'installation ne doit pas prendre plus de 10 minutes.
- Configurer une machine Linux pour le développement multiplateforme en C++
  - Visual Studio ne nécessite pas de distribution particulière de Linux. Le système d'exploitation peut s'exécuter sur un ordinateur physique, dans une machine virtuelle ou dans le cloud. Vous pouvez également utiliser le Sous-système Windows pour Linux (WSL). Toutefois, pour ce tutoriel, un environnement graphique est requis. WSL n'est pas recommandé ici, car il est principalement destiné aux opérations de ligne de commande.

- Visual Studio nécessite ces outils sur la machine Linux : compilateurs C++, gdb, ssh, rsync, make et zip. Sur les systèmes Debian, vous pouvez utiliser cette commande pour installer les dépendances suivantes :

  Invite de commandes Windows

  ```
  sudo apt install -y openssh-server build-essential gdb rsync make
  zip
  ```

- Visual Studio nécessite une version récente de CMake sur l'ordinateur Linux pour lequel le mode serveur est activé (au moins 3.8). Microsoft fournit une build universelle de CMake que vous pouvez installer sur n'importe quelle distribution Linux. Nous vous recommandons d'utiliser cette build pour vous assurer que vous disposez des dernières fonctionnalités. Vous pouvez obtenir les fichiers binaires de CMake à partir de la duplication (fork) Microsoft dans le dépôt CMake sur GitHub. Accédez à cette page et téléchargez la version qui correspond à l'architecture système sur votre machine Linux, puis marquez-la en tant qu'exécutable :

  Invite de commandes Windows

  ```
  wget <path to binary>
  chmod +x cmake-3.11.18033000-MSVC_2-Linux-x86_64.sh
  ```

- Vous pouvez voir les options d'exécution du script en spécifiant `--help`. Nous vous recommandons d'utiliser l'option `-prefix` pour spécifier l'installation dans le chemin **/usr** , car **/usr/bin** est l'emplacement par défaut où Visual Studio recherche CMake. L'exemple suivant montre le script Linux-x86_64. Modifiez-le en fonction des besoins si vous utilisez une autre plateforme cible.

  Invite de commandes Windows

  ```
  sudo ./cmake-3.11.18033000-MSVC_2-Linux-x86_64.sh --skip-license --
  prefix=/usr
  ```

- Git pour Windows installé sur votre machine Windows.
- Un compte GitHub.

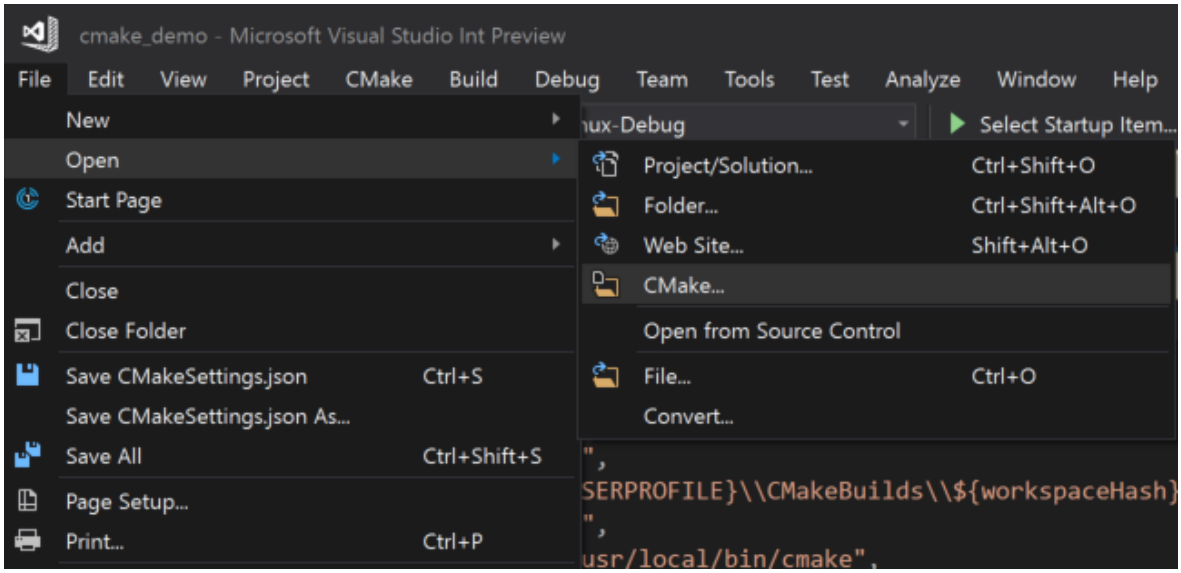
Cloner un projet CMake open source à partir de GitHub

Ce tutoriel utilise le Kit de développement logiciel (SDK) Bullet Physics sur GitHub. Il fournit des simulations physiques et de détection de collision pour de nombreuses applications. Le Kit de développement logiciel (SDK) inclut des exemples de programmes exécutables qui compilent et s'exécutent sans avoir à écrire de code supplémentaire. Ce tutoriel ne modifie pas le code source ou les scripts de build. Pour commencer, clonez le dépôt *bullet3* à partir de GitHub sur l'ordinateur sur lequel Visual Studio est installé.

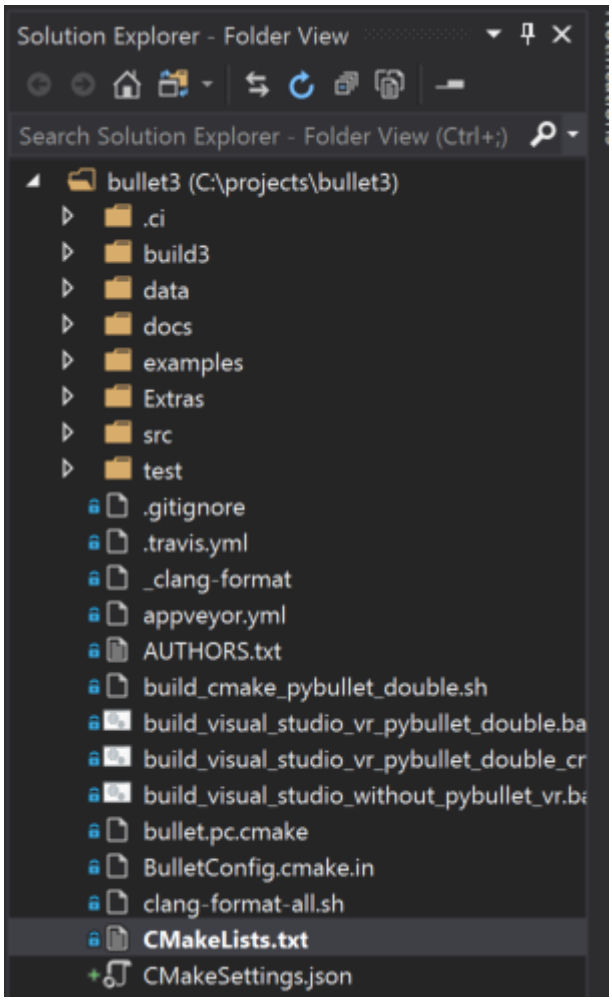
Invite de commandes Windows

```
git clone https://github.com/bulletphysics/bullet3.git
```

1. Dans le menu principal de Visual Studio, choisissez **Fichier** > **Ouvrir** > **CMake**. Accédez au fichier CMakeLists.txt à la racine du dépôt bullet3 que vous venez de télécharger.

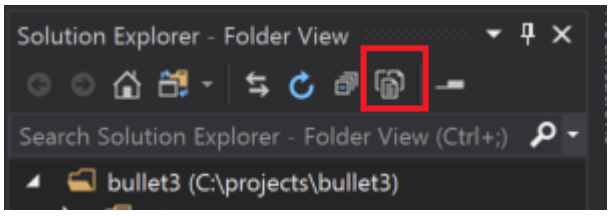


   Dès que vous ouvrez le dossier, la structure de votre dossier devient visible dans le **Explorateur de solutions**.



Cette vue montre le contenu réel du disque ; ce n'est pas une vue logique ou filtrée. Par défaut, elle n'affiche pas les fichiers masqués.

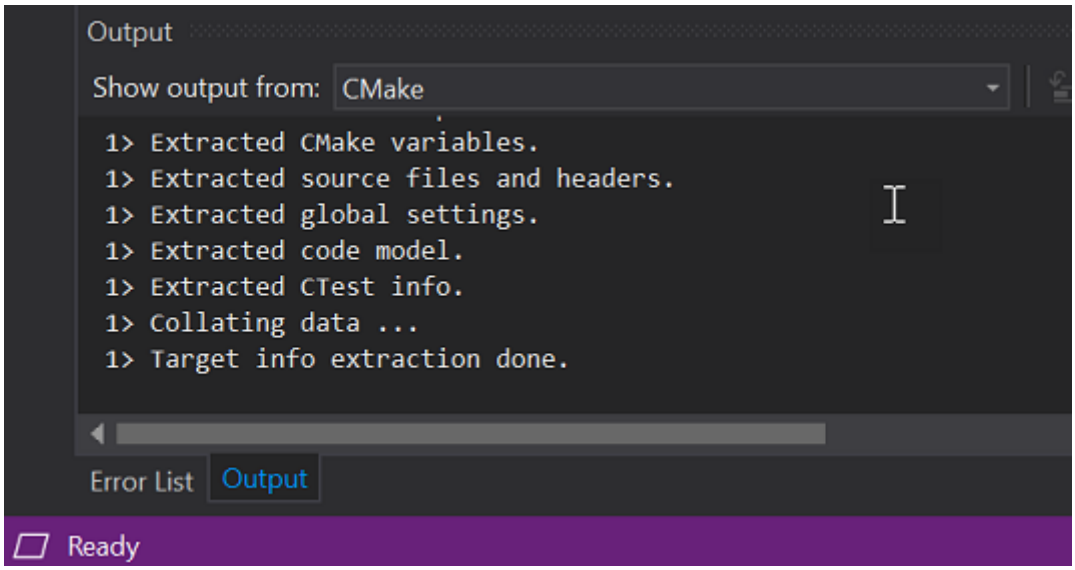
2. Choisissez le bouton **Afficher tous les fichiers** pour afficher tous les fichiers du dossier.



Basculer vers une vue des cibles

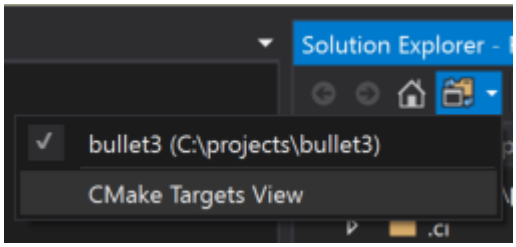
Quand vous ouvrez un dossier qui utilise CMake, Visual Studio génère automatiquement le cache CMake. Cette opération peut prendre plus ou moins de temps selon la taille de votre projet.

1. Dans la **fenêtre Sortie**, sélectionnez **Afficher la sortie à partir de**, puis choisissez **CMake** pour superviser l'état du processus de génération du cache. À la fin de l'opération, le message « Extraction des informations cibles terminée » s'affiche.

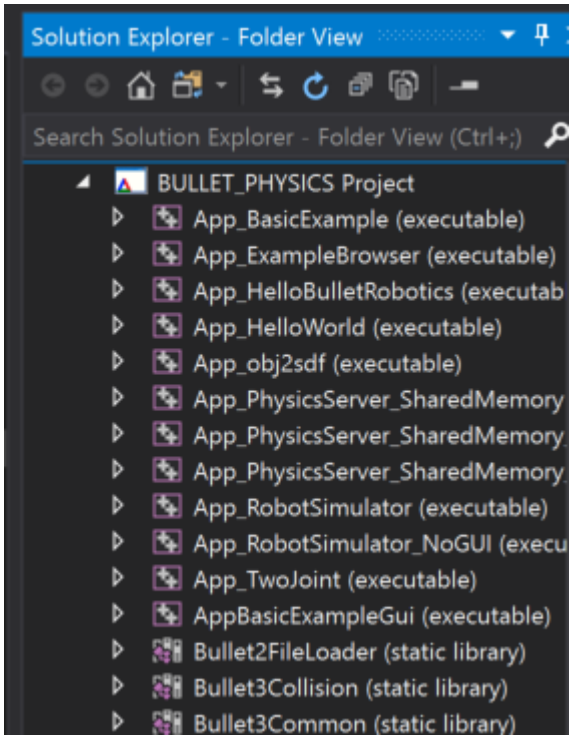


Une fois cette opération terminée, IntelliSense est configuré. Vous pouvez générer le projet et déboguer l'application. Visual Studio affiche désormais une vue logique de la solution, en fonction des cibles spécifiées dans les fichiers CMakeLists.

2. Utilisez le bouton **Solutions et dossiers** dans l'**Explorateur de solutions** pour passer à la vue des cibles de CMake.



Voici comment se présente cette vue pour le SDK Bullet :



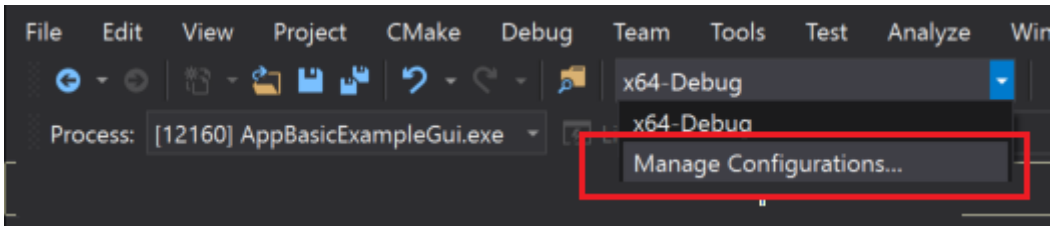
La vue des cibles offre une vue plus intuitive du contenu de cette base de code source. Vous pouvez voir que certaines cibles sont des bibliothèques alors que d'autres sont des exécutables.

3. Développez un nœud dans la vue des cibles de CMake pour voir les fichiers de code source associés, quel que soit leur emplacement sur le disque.

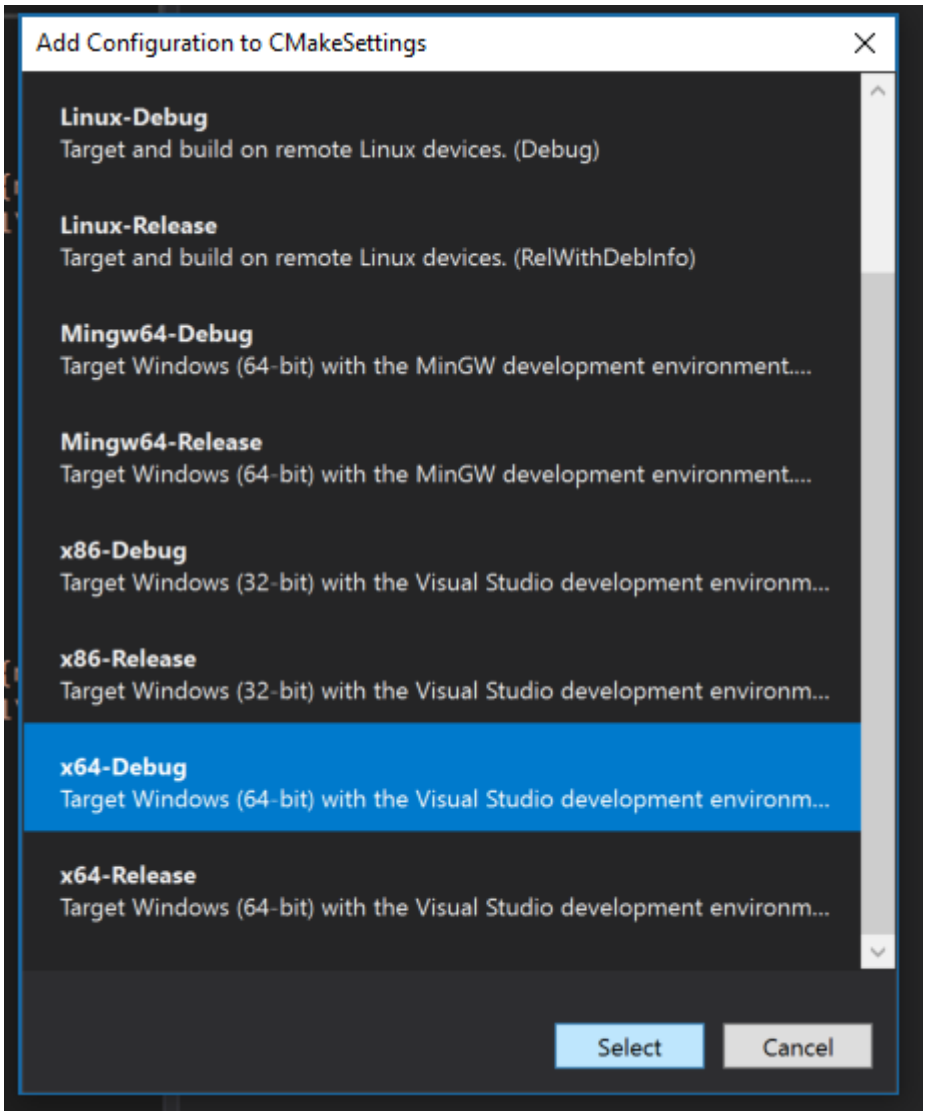
Ajouter une configuration x64-Debug explicite pour Windows

Visual Studio crée une configuration de **débogage x64** par défaut pour Windows. Les configurations indiquent à Visual Studio quelle plateforme cible utiliser pour CMake. La configuration par défaut n'est pas représentée sur le disque. Lorsque vous ajoutez explicitement une configuration, Visual Studio crée un fichier appelé *CMakeSettings.json*. Il est rempli avec les paramètres pour toutes les configurations que vous spécifiez.

1. Ajoutez une nouvelle configuration. Ouvrez la liste déroulante **Configuration** dans la barre d'outils, puis sélectionnez **Gérer les configurations**.



L'éditeur de paramètres CMake s'ouvre. Sélectionnez le signe plus vert sur le côté gauche de l'éditeur pour ajouter une nouvelle configuration. La boîte **de dialogue Ajouter une configuration à CMakeSettings** s'affiche.



Cette boîte de dialogue affiche toutes les configurations incluses dans Visual Studio, ainsi que toutes les configurations personnalisées que vous créez. Si vous souhaitez continuer à utiliser une configuration **x64-Debug** , il doit s'agir de la première que vous ajoutez. Sélectionnez **x64-Debug**, puis choisissez le bouton **Sélectionner** . Visual Studio crée le fichier CMakeSettings.json avec une configuration pour **x64-Debug** et l'enregistre sur le disque. Vous pouvez utiliser un nom de votre choix pour vos configurations en changeant le paramètre de nom directement dans CMakeSettings.json.

Définir un point d'arrêt, générer et exécuter sur Windows

Dans cette étape, nous allons déboguer un exemple de programme qui utilise la bibliothèque Bullet Physics.

1. Dans l'**Explorateur de solutions**, sélectionnez AppBasicExampleGui et développez-le.

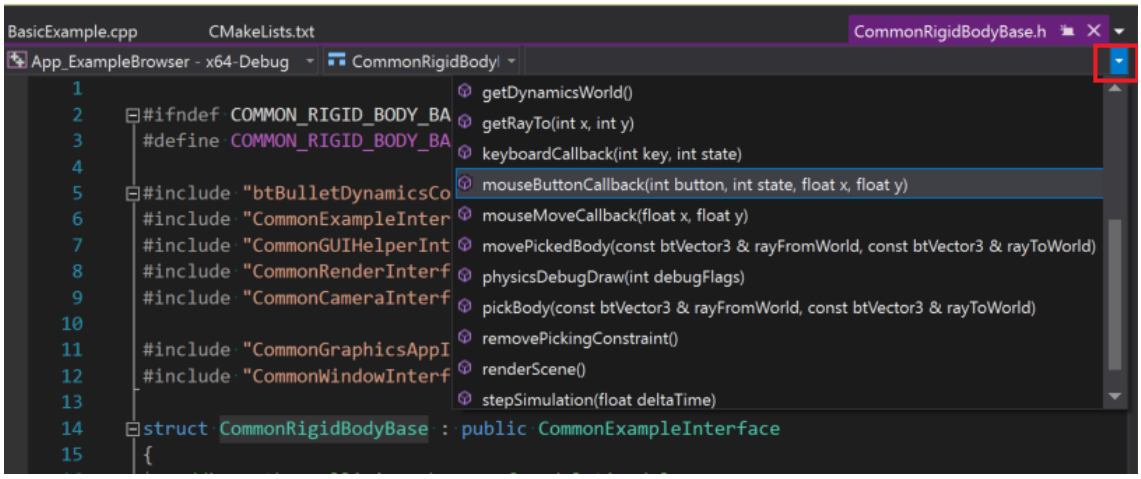
2. Ouvrez le fichier `BasicExample.cpp`.

3. Définissez un point d'arrêt qui est atteint lorsque vous cliquez dans l'application en cours d'exécution. L'événement de clic est géré dans une méthode au sein d'une classe d'assistance. Pour y accéder rapidement :

   a. Sélectionnez `CommonRigidBodyBase` dont le struct `BasicExample` est dérivé. C'est autour de la ligne 30.

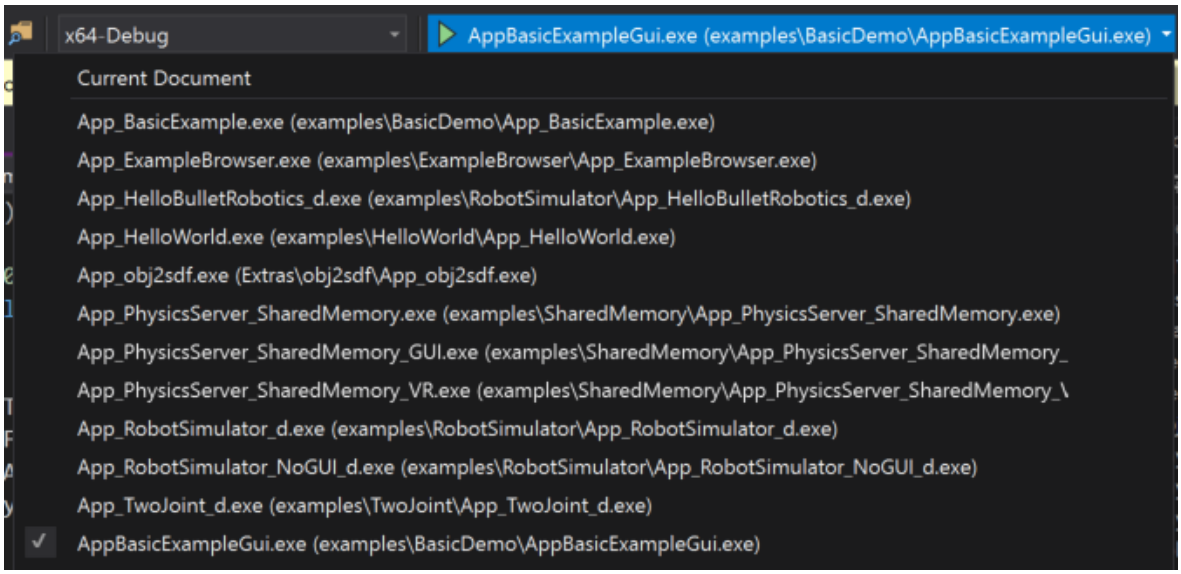
   b. Cliquez avec le bouton droit et choisissez **Atteindre la définition**. Vous êtes maintenant dans l'en-tête CommonRigidBodyBase.h.

   c. Dans l'affichage du navigateur au-dessus de votre source, vous devez voir que vous êtes dans le `CommonRigidBodyBase`. À droite, vous pouvez sélectionner les membres à examiner. Ouvrez la liste déroulante et sélectionnez `mouseButtonCallback` pour accéder à la définition de cette fonction dans l'en-tête.

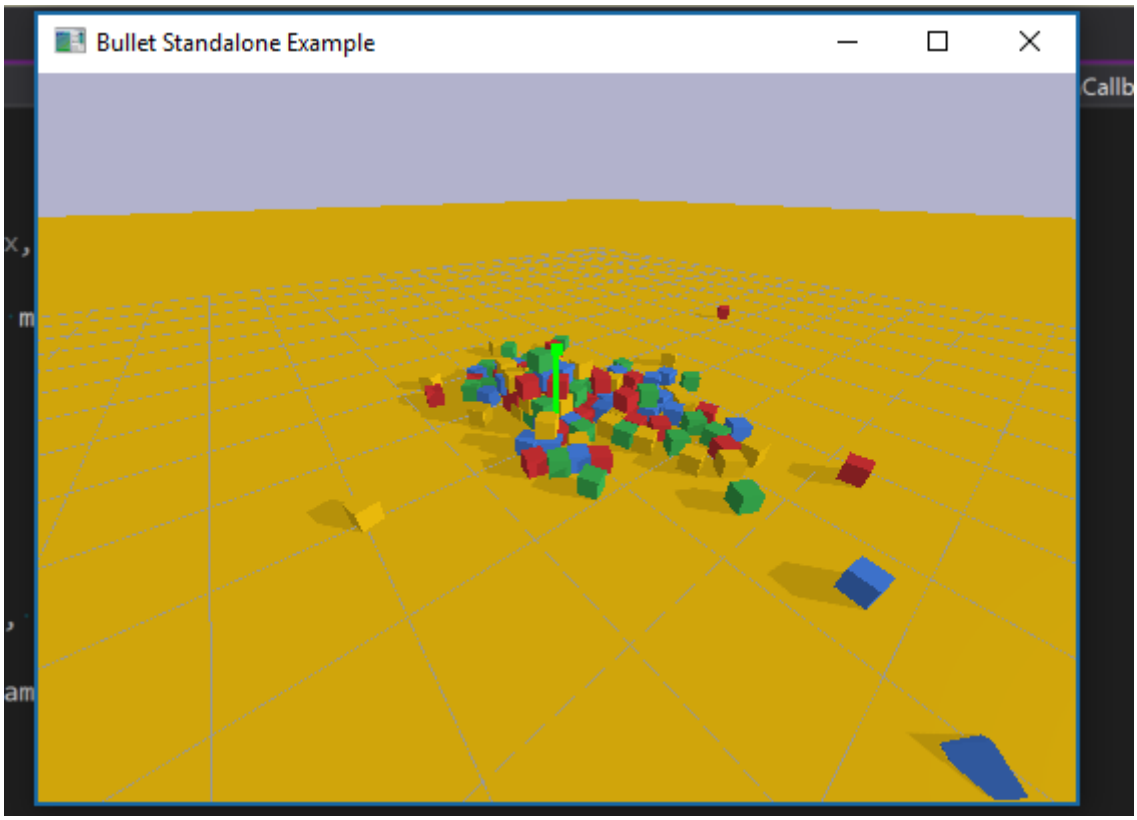


4. Ajoutez un point d'arrêt sur la première ligne dans cette fonction. Il est atteint lorsque vous cliquez sur un bouton de la souris dans la fenêtre de l'application, lors de l'exécution sous le débogueur Visual Studio.

5. Pour lancer l'application, sélectionnez la liste déroulante lancement dans la barre d'outils. Il s'agit de celui avec l'icône de lecture verte qui indique « Sélectionner l'élément de démarrage ». Dans la liste déroulante, sélectionnez AppBasicExampleGui.exe. Le nom de l'exécutable figure maintenant sur le bouton de lancement :



6. Choisissez le bouton de lancement pour générer l'application et les dépendances nécessaires, puis lancez-le avec le débogueur Visual Studio attaché. Après quelques instants, l'application exécutée apparaît :



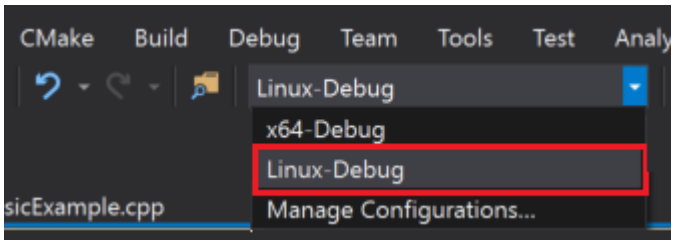
7. Déplacez votre souris dans la fenêtre d'application, puis cliquez sur un bouton pour déclencher le point d'arrêt. Le point d'arrêt ramène Visual Studio au premier plan, et l'éditeur montre la ligne où l'exécution est suspendue. Vous pouvez inspecter les variables d'application, les objets, les threads et la mémoire, ou parcourir votre code de manière interactive. Choisissez **Continuer** pour laisser l'application reprendre, puis quittez-la normalement. Ou bien, arrêtez l'exécution dans Visual Studio à l'aide du bouton Arrêter.

Ajouter une configuration Linux et se connecter à la machine distante

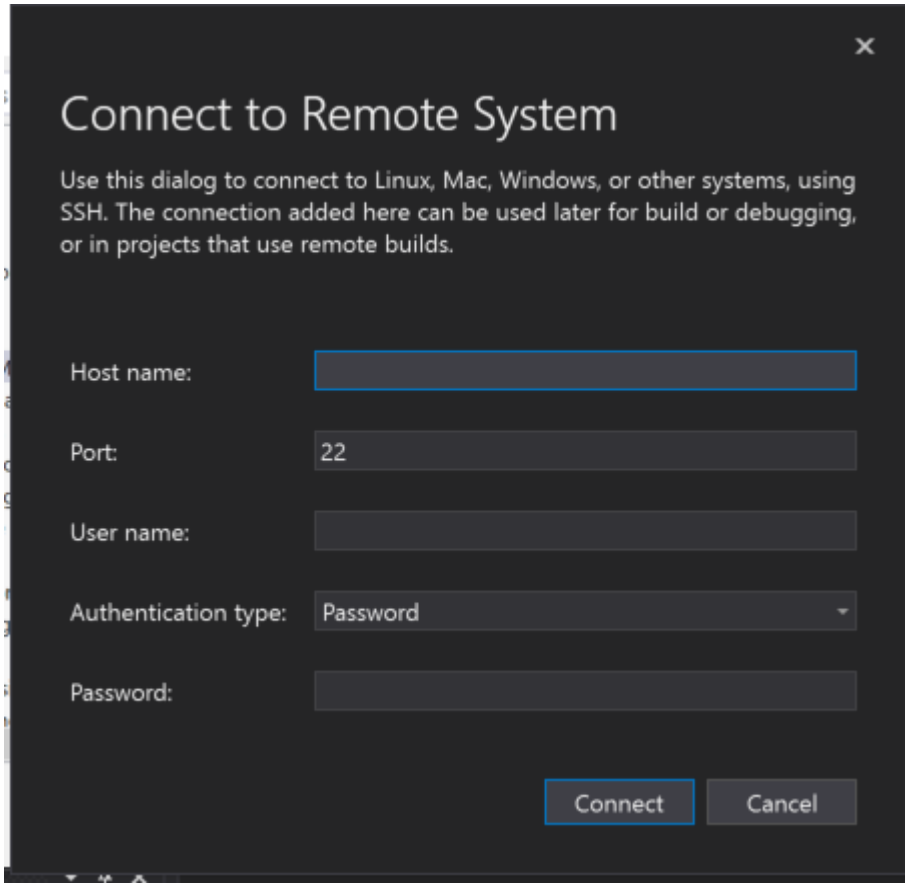
1. Ajoutez une configuration Linux. Cliquez avec le bouton droit sur le fichier CMakeSettings.json dans la vue de l'**Explorateur de solutions**, puis sélectionnez **Ajouter une configuration**. Comme précédemment, la boîte de dialogue Ajouter la configuration à CMakeSettings s'affiche. Sélectionnez **Linux-Déboguer** cette fois, puis enregistrez le fichier CMakeSettings.json (ctrl+ s).

2. **Visual Studio 2019 version 16.6 ou ultérieure** Faites défiler jusqu'au bas de l'éditeur de paramètres CMake, puis sélectionnez **Afficher les paramètres avancés**. Sélectionnez **Makefiles Unix** comme **générateur CMake**, puis enregistrez le fichier CMakeSettings.json (ctrl + s).

3. Sélectionnez **Linux-Déboguer** dans la liste déroulante de configuration.



Si c'est la première fois que vous vous connectez à un système Linux, la boîte de dialogue **Se connecter au système distant** s'affiche.



Si vous avez déjà ajouté une connexion à distance, vous pouvez ouvrir cette fenêtre en accédant à **Options** > **d'outils** > **multiplateforme** > **Gestionnaire des connexions**.

4. Fournissez les informations de connexion à votre machine Linux et choisissez **Se connecter**. Visual Studio ajoute cette machine à CMakeSettings.json comme connexion par défaut pour **Linux-Debug**. Il extrait également les en-têtes de votre ordinateur distant, de sorte que vous obtenez IntelliSense spécifique à cette connexion distante. Ensuite, Visual Studio envoie vos fichiers à l'ordinateur distant et génère le cache CMake sur le système distant. Ces étapes peuvent prendre un certain temps, en fonction de la vitesse de votre réseau et de la puissance de votre ordinateur distant. Vous saurez qu'il est terminé lorsque le message « Extraction des informations cibles terminée » s'affiche dans la fenêtre de sortie CMake.

Définir un point d'arrêt, générer et exécuter sur Linux

Comme il s'agit d'une application de bureau, vous devez fournir des informations de configuration supplémentaires à la configuration de débogage.

1. Dans la vue Cibles CMake, cliquez avec le bouton droit sur AppBasicExampleGui et choisissez **Paramètres de débogage et de lancement** pour ouvrir le fichier launch.vs.json qui se trouve dans le sous-dossier **.vs** masqué. Ce fichier est propre à votre environnement de développement local. Vous pouvez le déplacer à la racine de votre projet si vous souhaitez le vérifier et le partager avec votre équipe. Dans ce fichier, une configuration a été ajoutée pour AppBasicExampleGui. Ces paramètres par défaut fonctionnent dans la plupart des cas, mais pas ici. Étant donné qu'il s'agit d'une application de bureau, vous devez fournir des informations supplémentaires pour lancer le programme afin de pouvoir le voir sur votre machine Linux.

2. Pour rechercher la valeur de la variable `DISPLAY` d'environnement sur votre machine Linux, exécutez la commande suivante :

   Invite de commandes Windows

   ```
   echo $DISPLAY
   ```

   Dans la configuration d'AppBasicExampleGui, il existe un tableau de paramètres, « pipeArgs ». Il contient une ligne : « ${debuggerCommand} ». Il s'agit de la commande qui lance gdb sur l'ordinateur distant. Visual Studio doit exporter

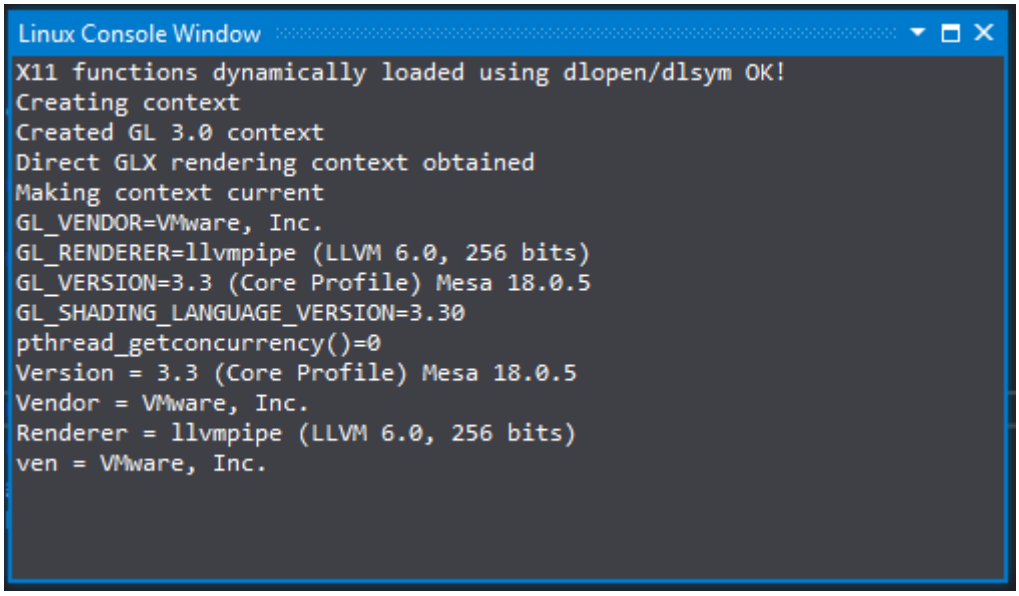
l'affichage dans ce contexte avant l'exécution de cette commande. Par exemple, si la valeur de votre affichage est `:1`, modifiez cette ligne comme suit :

Invite de commandes Windows

```
"export DISPLAY=:1;${debuggerCommand}",
```

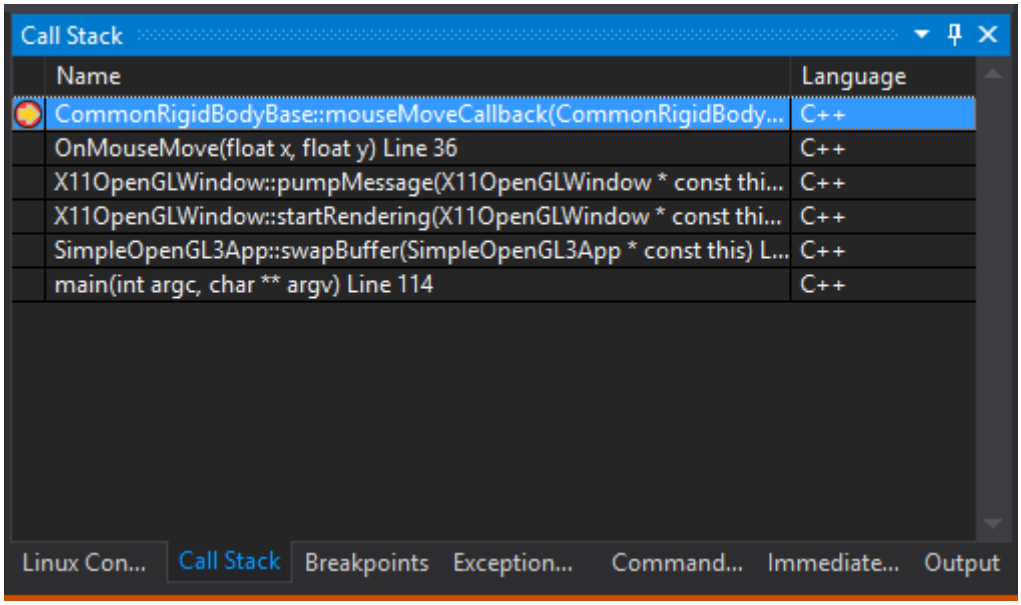
3. Lancez et déboguez votre application. Ouvrez la liste déroulante **Sélectionner l'élément de démarrage** dans la barre d'outils et choisissez **AppBasicExampleGui**. Ensuite, choisissez l'icône de lecture verte dans la barre d'outils ou appuyez sur **F5**. L'application et ses dépendances sont générées sur la machine Linux distante, puis lancées avec le débogueur Visual Studio attaché. Sur votre machine Linux distante, vous devez voir apparaître une fenêtre d'application.

4. Déplacez votre souris dans la fenêtre de l'application, puis cliquez sur un bouton. Le point d'arrêt est atteint. L'exécution du programme s'interrompt, Visual Studio revient au premier plan et vous voyez votre point d'arrêt. Vous devez également voir une fenêtre de console Linux s'ouvrir dans Visual Studio. La fenêtre fournit la sortie de l'ordinateur Linux distant et peut également accepter une entrée pour `stdin`. Comme n'importe quelle fenêtre Visual Studio, vous pouvez l'ancrer là où vous préférez la voir. Sa position est persistante dans les sessions ultérieures.



5. Vous pouvez alors inspecter les variables, les objets, les threads et la mémoire de l'application, et exécuter votre code pas à pas de manière interactive à l'aide de Visual Studio. Mais cette fois, vous effectuez tout cela sur une machine Linux distante au lieu de votre environnement Windows local. Vous pouvez choisir **Continuer** pour laisser l'application reprendre et quitter normalement, ou vous pouvez choisir le bouton Arrêter, comme avec l'exécution locale.

6. Si vous examinez la fenêtre Pile des appels, vous voyez les appels à `x11OpenGLWindow` puisque Visual Studio a lancé l'application sur Linux.



Ce que vous avez appris

Dans ce tutoriel, vous avez cloné une base de code directement à partir de GitHub. Vous l'avez créé, exécuté et débogué sur Windows sans modification. Ensuite, vous avez utilisé la même base de code, avec des modifications de configuration mineures, pour générer, exécuter et déboguer sur une machine Linux distante.

Étapes suivantes

Consultez les rubriques suivantes pour en savoir plus sur la configuration et le débogage des projets CMake dans Visual Studio :

Projets CMake dans Visual Studio

Configurer un projet CMake Linux

Se connecter à un ordinateur Linux distant

Personnaliser des paramètres de génération CMake

Configurer des sessions de débogage CMake

Déployer, exécuter et déboguer un projet Linux

Informations de référence sur la configuration prédéfinie de CMake

Configurer des projets Linux pour utiliser Address Sanitizer

Article • 16/06/2023

Dans Visual Studio 2019 version 16.1, la prise en charge d'AddressSanitizer (ASan) est intégrée aux projets Linux. Vous pouvez activer ASan pour les projets Linux basés sur MSBuild et pour les projets CMake. Cet outil fonctionne sur les systèmes Linux distants et sur le sous-système Windows pour Linux (WSL).

À propos d'ASan

ASan est un détecteur d'erreur de mémoire de runtime pour C/C++ qui intercepte les erreurs suivantes :

- Utilisation après libération (référence de pointeur non résolue)
- Dépassement de mémoire tampon au niveau du tas
- Dépassement de mémoire tampon au niveau de la pile
- Utilisation après retour
- Utilisation après la portée
- Bogues liés à l'ordre d'initialisation

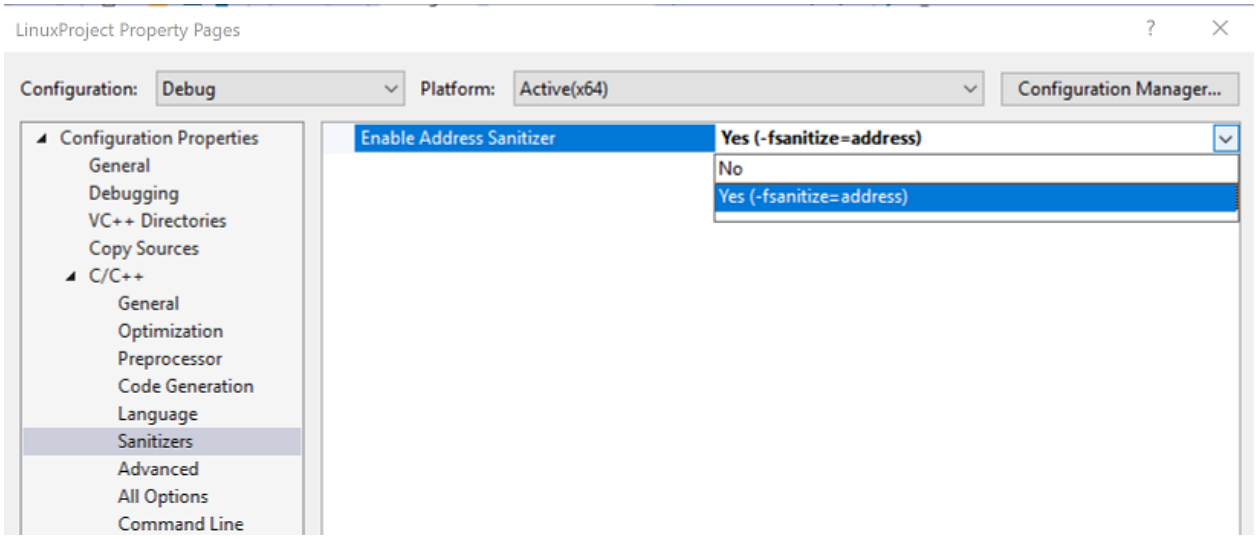
Quand ASan détecte une erreur, il arrête l'exécution immédiatement. Si vous exécutez un programme prenant en charge ASan dans le débogueur, vous voyez un message qui décrit le type d'erreur, l'adresse mémoire et l'emplacement dans le fichier source où l'erreur s'est produite :

Vous pouvez également voir la sortie ASan complète (y compris l'endroit où la mémoire endommagée a été alloué ou désallouée) dans le volet de débogage de la fenêtre de sortie.

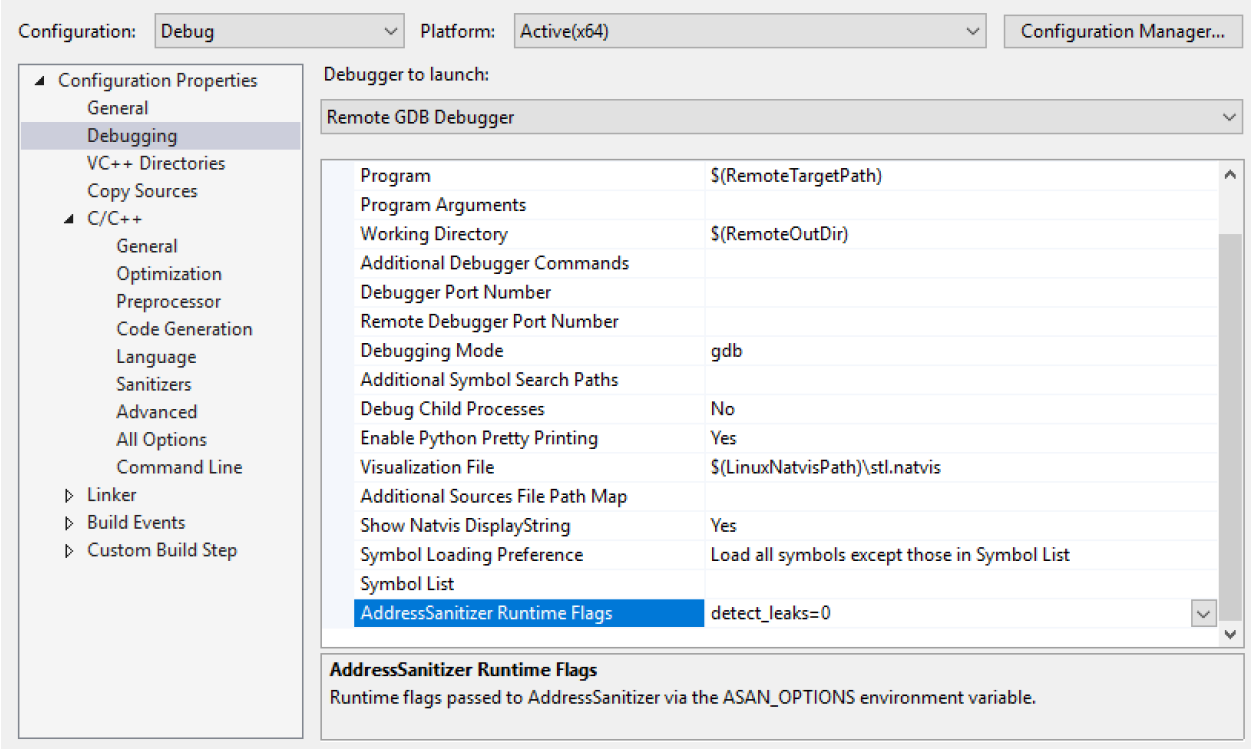
Activer ASan pour les projets Linux basés sur MSBuild

> **Notes**
>
> À compter de Visual Studio 2019 version 16.4, AddressSanitizer pour les projets Linux est activé via **Propriétés de projet> Propriétés de configuration>C/C++>Activer l'assainisseur d'adresse**.

Pour activer ASan pour les projets Linux basés sur MSBuild, cliquez avec le bouton droit sur le projet dans l'**Explorateur de solutions**, puis sélectionnez **propriétés**. Ensuite, accédez à **Propriétés de configuration>C/C++>Détecteurs d'erreurs**. Activé par le biais d'indicateurs de compilateur et d'éditeur de liens, ASan requiert que votre projet soit recompilé pour fonctionner.



Vous pouvez passer des indicateurs de runtime ASan facultatifs en accédant à **Propriétés de configuration>Débogage>Indicateurs d'exécution AddressSanitizer**. Cliquez sur la flèche vers le bas pour ajouter ou supprimer des indicateurs.

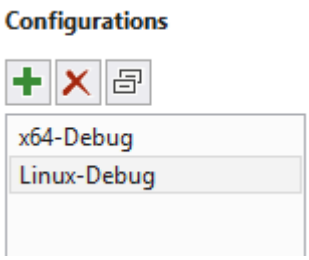


Activer ASan pour les projets CMake Visual Studio

> ⓘ Notes
>
> Pour générer avec des présélections CMake, commencez par activer ASan dans votre fichier CMakeLists.txt. Pour plus d'informations, consultez **Activer AddressSanitizer pour Windows et Linux**.

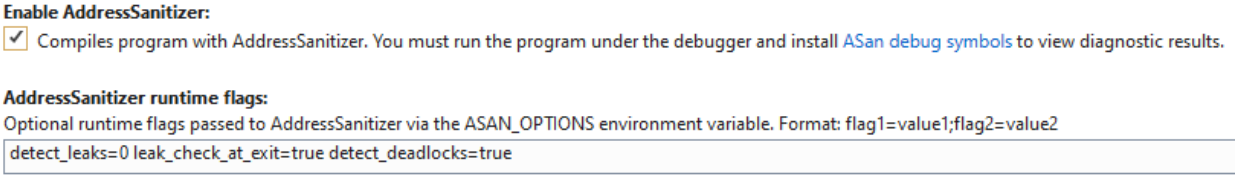
Pour activer ASan pour CMake, cliquez avec le bouton droit sur le fichier `CMakeLists.txt` dans l'**Explorateur de solutions**, puis choisissez **Paramètres CMake du projet**.

Vérifiez qu'une configuration Linux (par exemple, **Linux-Debug**) est sélectionnée dans le volet gauche de la boîte de dialogue :



Les options d'ASan se trouvent sous **Général**. Entrez les indicateurs de runtime ASan au format « indicateur=valeur », en les séparant par des espaces. L'interface utilisateur

suggère erronément d'utiliser des points-virgules. Utilisez des espaces ou des points-virgules pour séparer les indicateurs.



Installer les symboles de débogage ASan

Pour activer les diagnostics ASan, vous devez installer ses symboles de débogage (libasan-dbg) sur votre machine Linux distante ou sur l'installation WSL. La version de libasan-dbg que vous chargez dépend de la version de GCC installée sur votre machine Linux :

Version d'ASan	Version de GCC
libasan0	gcc-4.8
libasan2	gcc-5
libasan3	gcc-6
libasan4	gcc-7
libasan5	gcc-8

Vous pouvez déterminer la version de GCC que vous avez à l'aide de cette commande :

Bash

```
gcc --version
```

Pour afficher la version de libasan-dbg dont vous avez besoin, exécutez votre programme, puis examinez le volet **Déboguer** de la fenêtre **Sortie**. La version d'ASan chargée correspond à la version de libasan-dbg nécessaire sur votre machine Linux. Vous pouvez utiliser **Ctrl+F** pour rechercher « libasan » dans la fenêtre. Si vous avez libasan4, par exemple, vous voyez une ligne comme celle-ci :

Output

```
Loaded '/usr/lib/x86_64-linux-gnu/libasan.so.4'. Symbols loaded.
```

Vous pouvez installer les bits de débogage ASan sur les distributions Linux qui utilisent apt avec la commande suivante. Cette commande installe la version 4 :

Bash

```
sudo apt-get install libasan4-dbg
```

Pour obtenir des instructions complètes sur l'installation des packages de symboles de débogage sur Ubuntu, consultez Packages de symboles de débogage.

Si ASan est activé, Visual Studio vous invite en haut du volet **Déboguer** de la fenêtre **Sortie** à installer les symboles de débogage ASan.