Christophe PICHAUD
Consultant sur les technologies Microsoft
christophepichaud@hotmail.com | www.windowscpp.com

Créer un service Windows

Partie 1

LE SQUELETTE TECHNIQUE

Sous Windows, nous développons des applications EXE ou des modules partagés DLL. Il existe une variante ou classe d'application que l'on nomme service Windows qui s'exécute en arrière-plan même si personne n'a ouvert de session sur la machine. Il peut être démarré en mode automatique, au boot de la machine, en mode manuel ou hors de service. Il existe deux types de services : des drivers et des services qui ne sont pas des drivers. C'est le type non-drivers que nous allons étudier ici.

Le SCM : Service Controler Manager

Les services sont gérés dans une base de données (la base de registres) et c'est le SCM, démarré au démarrage de la machine, qui se charge des fonctionnalités suivantes :

- Gestion de la base des services ;
- Démarrage des services au démarrage du système ou à la demande ;
- Gestion des informations de statut de tous les types de services ;
- Verrouillage de la base des services.

La base de données du SCM est dans la base de registres sous : HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services
Malgré le fait que la base de registres puisse être modifiée à la main, nous allons passer par les API dédiées aux services Windows. Pour créer un service, il faut utiliser la fonction Win32 CreateService :

```
SC_HANDLE WINAPI CreateService(
  _In_      SC_HANDLE hSCManager,
  _In_      LPCTSTR   lpServiceName,
  _In_opt_  LPCTSTR   lpDisplayName,
  _In_      DWORD     dwDesiredAccess,
  _In_      DWORD     dwServiceType,
  _In_      DWORD     dwStartType,
  _In_      DWORD     dwErrorControl,
  _In_opt_  LPCTSTR   lpBinaryPathName,
  _In_opt_  LPCTSTR   lpLoadOrderGroup,
  _Out_opt_ LPDWORD   lpdwTagId,
  _In_opt_  LPCTSTR   lpDependencies,
  _In_opt_  LPCTSTR   lpServiceStartName,
  _In_opt_  LPCTSTR   lpPassword
);
```

Vous remarquerez que le premier paramètre est un handle sur le SCM. Pourquoi ? En fait, il est nécessaire d'ouvrir le SCM avant d'utiliser cette fonction. Voici le code de création d'un service :

```
bool CServiceModule::Install()
{
  if (IsInstalled())
  {
    std::wcout << _T("Service is already installed") << std::endl;
    return TRUE;
  }

  SC_HANDLE hSCM = ::OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
  if (hSCM == NULL)
  {
    std::wcout << _T("Couldn't open service manager") << std::endl;
    return FALSE;
  }

  // Get the executable file path
  TCHAR szFilePath[_MAX_PATH];
  ::GetModuleFileName(NULL, szFilePath, _MAX_PATH);

  SC_HANDLE hService = ::CreateService(
    hSCM,
    SERVICE_NAME,
    SERVICE_NAME,
    SERVICE_ALL_ACCESS, SERVICE_WIN32_OWN_PROCESS,
    SERVICE_DEMAND_START, SERVICE_ERROR_NORMAL,
    szFilePath, NULL, NULL, NULL, NULL, NULL);

  if (hService == NULL)
  {
    ::CloseServiceHandle(hSCM);
    std::wcout << _T("Couldn't create service ") << SERVICE_NAME << std::endl;
    return FALSE;
  }

  ::CloseServiceHandle(hService);
  ::CloseServiceHandle(hSCM);
  return TRUE;
}
```

La fonction possède quelques subtilités que je n'expliquerai pas ici mais cherchez dans le MSDN Library si vous êtes curieux. Il est ainsi possible de créer un service qui dépend d'un autre service, etc. La routine IsInstalled() est appelée pour que le code soit propre :

```
bool CServiceModule::IsInstalled()
{
  bool bResult = false;

  SC_HANDLE hSCM = ::OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);

  if (hSCM != NULL)
  {
```

```
    SC_HANDLE hService = ::OpenService(hSCM,
                        SERVICE_NAME,
                        SERVICE_QUERY_CONFIG);
    if (hService != NULL)
    {
        bResult = true;
        ::CloseServiceHandle(hService);
    }
    ::CloseServiceHandle(hSCM);
  }
  return bResult;
}
```

Pour que ce code soit exécuté correctement, il faut être en mode administrateur. Voici le code du wmain qui permet de créer le service si on lui passe le fag « -createservice »:

```
CServiceModule _Module;

int wmain(int argc, wchar_t *argv[])
{
  _Module.m_bService = TRUE;
  std::wstring mode;

  if (argc == 2)
  {
      mode = argv[1];
  }

  if (mode == _T("-console"))
  {
      std::wcout << _T("Running in console mode.") << std::endl;
      _Module.m_bService = FALSE;
      _Module.Start();
      return 0;
  }

  if (mode == _T("-createservice"))
  {
      std::wcout << _T("Creating the service...") << std::endl;

      CServiceModule sm;
      sm.Install();
      return 0;
  }

  _Module.m_bService = TRUE;
  _Module.Start();

  // When we get here, the service has been stopped
  return _Module.m_status.dwWin32ExitCode;
}
```

Lançons le code en passant le paramètre –createservice : 1
Pour gérer les services Windows, il existe un outil sc.exe. Il est aussi possible de créer, détruire, lister des services juste avec cet outil sc. Voyons la base de registres où le SCM gère les services et cherchons la clé « MyService » : 2. Maintenant que le service est créé, il ne nous reste plus qu'à lui envoyer une commande de type :

- net start MyService
- net stop MyService

Vous allez me dire, on fait start de quoi ? Oui c'est vrai, il nous faut le code qui gère cette commande. Le SCM va nous transmettre les ordres de START ou de STOP (il existe aussi PAUSE). Il faut donc s'enregistrer auprès du SCM pour gérer ces évènements. Dans le wmain(), on appelle une méthode Start() :

```
void CServiceModule::Start()
{
  SERVICE_TABLE_ENTRY st[] =
  {
     { SERVICE_NAME, _ServiceMain },
     { NULL, NULL }
  };

  if (m_bService && !::StartServiceCtrlDispatcher(st))
  {
     m_bService = FALSE;
  }
  if (m_bService == FALSE)
  {
     Run();
  }
}
```

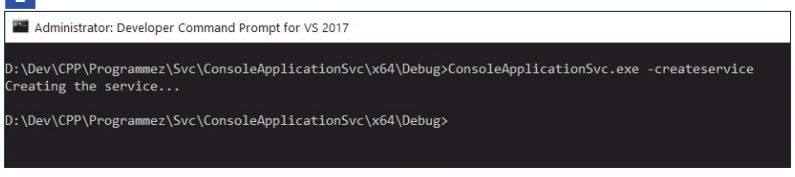
La fonction Win32 StartServiceCtrlDispatcher() permet de se connecter au SCM en lui fournissant le point d'entrée du service ; ici : _ServiceMain() :

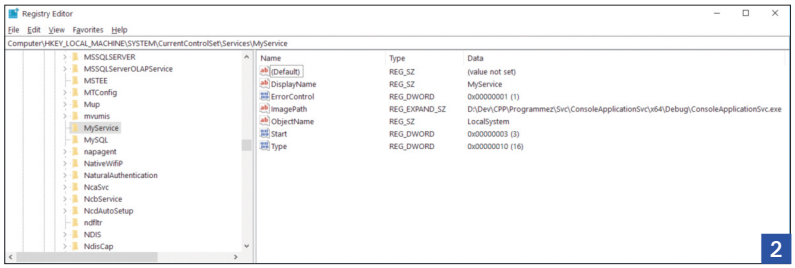
```
void CServiceModule::_ServiceMain(DWORD dwArgc, LPTSTR * lpszArgv)
{
  _Module.ServiceMain(dwArgc, lpszArgv);
}
```

Nous donnons la main à l'instance du module qui est une variable globale.

```
void CServiceModule::ServiceMain(DWORD dwArgc, LPTSTR * lpszArgv)
{
  m_status.dwCurrentState = SERVICE_START_PENDING;
  m_hServiceStatus = RegisterServiceCtrlHandler(SERVICE_NAME, _Handler);
  if (m_hServiceStatus == NULL)
```

1





2

```
  {
    //LogEvent(_T("Handler not installed"));
    return;
  }
  SetServiceStatus(SERVICE_START_PENDING);

  m_status.dwWin32ExitCode = S_OK;
  m_status.dwCheckPoint = 0;
  m_status.dwWaitHint = 0;

  // When the Run function returns, the service has stopped.
  Run();

  SetServiceStatus(SERVICE_STOPPED);
  //LogEvent(_T("Service stopped"));
  //_Trace.LogDebug("CServiceModule::ServiceMain", "Service stopped");

}
```

Ici, nous faisons une autre liaison avec le SCM en utilisant la fonction RegisterServiceCtrlHandler et nous appelons Run() :

```
void WINAPI CServiceModule::_Handler(DWORD dwOpcode)
{
  _Module.Handler(dwOpcode);
}

void CServiceModule::Handler(DWORD dwOpcode)
{
  switch (dwOpcode)
  {
  case SERVICE_CONTROL_STOP:
    m_bStop = TRUE;
    SetServiceStatus(SERVICE_PAUSED);
    break;
  case SERVICE_CONTROL_PAUSE:
    break;
  case SERVICE_CONTROL_CONTINUE:
    break;
  case SERVICE_CONTROL_INTERROGATE:
    break;
  case SERVICE_CONTROL_SHUTDOWN:
    break;
  default:
    //LogEvent(_T("Bad service request"));
    break;
  }
}
```

Ici, on gère les états du service. Il faut absolument gérer le stop. Voyons le corps de méthode Run() dont nous avons parlé :

```
void CServiceModule::Run()
{
  _Module.dwThreadID = GetCurrentThreadId();

  //LogEvent(_T("Service started"));
  //_Trace.LogError("CServiceModule::Run", "Service started");

  if (m_bService)
    SetServiceStatus(SERVICE_RUNNING);

  if (m_bService)
  {
    //CString str = "AfxBeginThread";

    m_pThread = AfxBeginThread(AutomateThread, 0, THREAD_PRIORITY_
NORMAL, CREATE_SUSPENDED, NULL);
    if (m_pThread == NULL)
    {
      //LogEvent("Creation du thread AutomateThread impossible");
      SetServiceStatus(SERVICE_STOPPED);
      goto quit_now;
      return;
    }
    m_pThread->ResumeThread();
  }
  else
  {
    //AfxMessageBox("Utilisez NET START pour lancer le service.");
    //PostThreadMessage(_Module.dwThreadID, WM_QUIT, 0, 0);
  }

  MSG msg;
  while (GetMessage(&msg, 0, 0, 0))
    DispatchMessage(&msg);

quit_now:
  return;
}
```

Cette méthode lance un thread : c'est le corps principal du service : un thread en arrière-plan. Dans ce thread, on y ajoute ce que l'on veut. Un serveur TCP/IP, une fabrique de composants COM, etc. Voici le code :

```
UINT AutomateThread(LPVOID pParam)
{
  CString strLog;
  try
  {
    CoInitialize(NULL);

    while (TRUE)
    {
      if (_Module.m_bStop)
      {
        goto stop_service;
        break;
      }
      Sleep(200);

    } // Main loop
```

```
  stop_service:
    strLog.Format(_T("Arrêt du service"));
    //_Module.LogEvent(strLog);
    _Module.SetServiceStatus(SERVICE_STOP_PENDING);
    PostThreadMessage(_Module.dwThreadID, WM_QUIT, 0, 0);
  }
  catch (...)
  {
    strLog.Format(_T("EXCEPTION: Arrêt du service"));
    //_Module.LogEvent(strLog);
    //_Trace.LogDebug("AutomateThread", strLog);
    _Module.SetServiceStatus(SERVICE_STOP_PENDING);
    PostThreadMessage(_Module.dwThreadID, WM_QUIT, 0, 0);
  }

  CoUninitialize();
  return 0;
}
```

On remarquera que les états du service sont explicitement déclarés dans le code via la fonction Win32 SetServiceStatus :

```
void CServiceModule::SetServiceStatus(DWORD dwState)
{
  m_status.dwCurrentState = dwState;
  ::SetServiceStatus(m_hServiceStatus, &m_status);
}
```

Dans notre exemple, le code est tiré d'un service qui fonctionne, c'est un ensemble de composants COM qui sont accessibles au sein de service Windows donc le thread ne fait rien si ce n'est d'attendre la notification du stop de service via une commande : net stop MyService. Pour être complet, affichons le code complet de la classe (le header) :

```
#pragma once

#define SERVICE_NAME _T("MyService")

class CServiceModule
{
public:
  CServiceModule();
  ~CServiceModule();

public:
  bool IsInstalled();
  bool Install();
  void Start();

public:
  void ServiceMain(DWORD dwArgc, LPTSTR * lpszArgv);
  void SetServiceStatus(DWORD dwState);
  void Handler(DWORD dwOpcode);
  void Run();

public:
  static void WINAPI _Handler(DWORD dwOpcode);
  static void WINAPI _ServiceMain(DWORD dwArgc, LPTSTR * lpszArgv);

public:
  SERVICE_STATUS_HANDLE m_hServiceStatus;
  SERVICE_STATUS m_status;
  bool m_bService;
  bool m_bStop;
  DWORD dwThreadID;
  CWinThread * m_pThread;
};

extern CServiceModule _Module;
UINT AutomateThread(LPVOID pParam);
```

On notera que le code est assez technique. D'un autre côté, une fois qu'on a fait un service, c'est toujours la même chose donc on peut réutiliser les classes et méthodes. Le nom du service est une constante.

Le logger système

Historiquement depuis Windows NT 3.51, il y a un service log des informations dans le journal des évènements. Vous pouvez remarquer que dans le code, il y a des _Module.LogEvent() en commentaires…

Conclusion

Construire un service nécessite un code technique, il faut l'avouer. Ce code est réutilisable donc le faire une fois permettra de le réutiliser. Dans cette partie on a vu comment faire la liaison avec le SCM et comment développer les points d'entrées. •