Reverse Engineering

Überarbeitung der Initialisierungsdateistruktur des XCTL-Programms zur Verbesserung der Sicherheit

von
Jens Klier

Inhalt

1 Motivation

Das XCTL-Programm steuert Motoren und Detektoren in einem Versuchsaufbau zur Röntgenstrukturanalyse. Dabei werden alle software- und hardwarespezifischen Parameter in einer Initialisierungsdatei abgespeichert. Diese Datei wird vom Programm verändert, kann aber auch mit Hilfe eines Texteditors bearbeitet werden. In dieser Dabei dürfen aber nur anwenderspezifische Parameter geändert werden. Ein versehentliches Ändern der hardwarespezifischen Parameter kann dazu führen, das die Motoren oder Detektoren beschädigt oder sogar zerstört werden. Meine Aufgabe bestand darin, die Initialisierungsdatei des XCTL-Programms zu analysieren, und eine Methode zu entwickeln, welche die sensiblen Daten vor unbeabsichtigter und beabsichtigter Veränderung schützt. Des weiteren sollte es zu einem späteren Zeitpunkt der Programmentwicklung möglich sein, anwenderspezifische Initialisierungsparameter in einer speziellen Initialisierungsdatei dem Programm zu übergeben. Arbeitsplatzspezifische Initialisierungsparameter sollen aber nicht verändert werden können. Es war also zu überprüfen, welche der Parameter arbeitsplatzspezifisch sind und diese dann in einer speziell geschützten Initialisierungsdatei zu sichern.

2 Analyse des Programmzustandes

2.1 Vorgehensweise und Analysemethoden

Das Hauptproblem dieser Aufgabe bestand darin, herauszufinden, wie der Zugriff auf die Initialisierungsdatei realisiert wurde und ob er im gesamten Projekt einheitlich verwendet wurde. Um Informationen über die Initialisierungsdateizugriffe zu erhalten, habe ich mich dazu entschlossen, den Quelltext zu analysieren. Dazu habe ich alle Quelldateien nach bestimmten Schlüsselwörtern durchsucht und mir dann die entsprechenden Stellen angesehen. Als Analysehilfe habe ich anstelle der Borland Entwicklungsumgebung die Visual C++ Entwicklungsumgebung von Microsoft benutzt. Dort kann das Programm zwar nicht kompiliert werden, aber eine Quelltextanalyse wird durchgeführt und graphisch aufbereitet. Außerdem ist die Suche nach Schlüsselwörtern über alle Projektdateien möglich, die gefundenen Dateien werden per Mausklick in den Editor geladen und können so problemlos eingesehen werden. Damit ist eine komfortable und schnelle Quelltextanalyse möglich.

2.2 Analyse der Initialisierungsdatei

Die Initialisierungsdatei des XCTL-Programmes besteht aus 10 Hauptsektionen, wobei es beliebig viele Motor- und Device-Sektionen geben kann.

- [Steuerprogramm]
- [ManualMoves]
- [Counter]
- [AreaScan]
- [AreaScanCCD]
- [Scan]
- [Topographie]
- [GPIB0]
- [Motorx] x=1..n
- [Devicex] x=1..n
- [MOTORSIM]

Die Sektionen [Steuerprogramm], [ManualMoves], [Counter], [AreaScan], [AreaScanCCD], [Scan] und [Topographie] enthalten ausschließlich anwenderspezifische Initialisierungsparameter, während die Sektionen [GPIB0], [Motorx] und [Devicex] ausschließlich arbeitsplatzspezifische Initialisierungsparameter enthalten. Eine genaue Beschreibung der einzelnen Parameter erfolgt in Anhang A.

2.3 Analyse der Zugriffsmethoden

Die Analyse der Quellfiles ergab, das der Zugriff auf die Initialisierungsdatei ausschließlich über die Methoden GetPrivateProfile... und WritePrivateProfile... durchgeführt wird.

Quelldateien	Zugriff auf die Sektionen
am9513.cpp	[Devicex] x=1..n
braunpsd.cpp	[Devicex] x=1..n
counters.cpp	[Devicex] x=1..n
motors.cpp	[GPIB0], [Motorx] x=1..n

m_arscan.cpp	[Devicex] x=1..n, [AreaScan]
m_data.cpp	[AreaScan]
m_device.cpp	[Counter]
m_layer.cpp	[Steuerprogramm]
m_main.cpp	[Steuerprogramm], [ManualMoves]
m_scan.cpp	[Scan]
m_topo.cpp	[Topography]
m_ccdscn.cpp	[AreaScanCCD]
msimstat.cpp	[MOTORSIM]

Tabelle 1: Quelldateien mit Zugriff auf die Initialisierungsdatei

Die Methoden Get/WritePrivateProfile... haben folgende Syntax :

```
BOOL WritePrivateProfileString(
   LPCTSTR lpAppName,        // pointer to section name
   LPCTSTR lpKeyName,        // pointer to key name
   LPCTSTR lpString,         // pointer to string to add
   LPCTSTR lpFileName        // pointer to initialization filename  );

DWORD GetPrivateProfileString(
   LPCTSTR lpAppName,        // points to section name
   LPCTSTR lpKeyName,        // points to key name
   LPCTSTR lpDefault,        // points to default string
   LPTSTR lpReturnedString,  // points to destination buffer
   DWORD nSize,              // size of destination buffer
   LPCTSTR lpFileName        // points to initialization filename   );

UINT GetPrivateProfileInt(
   LPCTSTR lpAppName,        // address of section name
   LPCTSTR lpKeyName,        // address of key name
   INT nDefault,             // return value if key name is not found
   LPCTSTR lpFileName        // address of initialization filename  );
```

Der Initialisierungsdateiname lpFileName wird ausschließlich durch die parameterlose Methode GetCFile() übergeben. Die Deklaration der Methode erfolgt in der Datei l_layer.cpp. Dort erfolgt auch die Deklaration und Initialisierung der Variablen MainP. Die Initialisierung erfolgt über die Methode InitializeSPLibrary().

```
LPCSTR _export WINAPI GetCFile(void)
   {
   return (LPCSTR)MainP.IniFile;  };
```

```
BOOL _export WINAPI InitializeSPLibrary(TMainParameters* MainParam)
   {
   MainP = *MainParam;
                                                    :
   };
```

Die Methode GetCFile() ermittelt den Namen der Initialisierungsdatei aus der Variablen MainP.IniFile. Diese ist Bestandteil einer Struktur und wird in der Datei comclass.h deklariert. Dort wird auch eine Variable IFile deklariert, in der der Initialisierungsdateiname zwischengespeichert wird. Beide Variablen werden als public in der Klasse TMain deklariert.

```
typedef struct
{
  HWND      hWndFrame;
  HWND      hWndClient;
  HINSTANCE hInstance;
  UINT      wm_DrawStatus;
  BOOL      bCreateIniDefaults;
  char      IniFile[MAXPATH];
} TMainParameters;


class TMain
{
 public:
                                                    :
  char            IFile[MAXPATH];
                                                    :
  TMainParameters  Parameters;
                                                    :
};
```

Die Ermittlung des Initialisierungsdateinamens erfolgt in mehreren Schritten in der Datei m_main.cpp.

1. Schritt

Im Konstruktor TMain() wird die globale Variable IFile mit dem aktuellen Path und dem Namen des Programms initialisiert. Dort erfolgt auch eine Überprüfung, ob die Datei vorhanden ist. Dementsprechend wird die globale Variable bNoConfigurationFile, die in der Datei m_main.cpp deklariert ist, auf TRUE oder FALSE gesetzt.

```
BOOL         bNoConfigurationFile;


TMain::TMain(void)
   {
                                                    :
   sprintf(IFile,"%s%s.ini",Main.szDirectory,Name);
   if(-1 == OpenFile(IFile,&of,OF_EXIST))
           bNoConfigurationFile = TRUE;
```

```
        else
        bNoConfigurationFile = FALSE;
                                                    :
    };
```

2. Schritt

In der Methode SetParameters() wird dann die Variable Parameters.IniFile mit der Variablen lFile initialisiert.

```
    void TMain::SetParameters(void)
        {
                                                    :
        strcpy(Parameters.IniFile,lFile);
        };
```

3. Schritt

Die Initialisierung der eigentlichen Struktur MainP, und damit der Variablen MainP.IniFile, erfolgt im Hauptprogramm in der Datei m_main.cpp über mehrere Methodenaufrufe. Zuerst wird die Methode FrameWndProc() aufgerufen, diese ruft die Methode DoCommandsFrame(), die wiederum die Methode InitializeSPLibrary() aufruft, welche letztendlich die Struktur MainP initialisiert.

```
    LRESULT CALLBACK _export FrameWndProc(HWND hWindow,UINT message,WPARAM
wParam,LPARAM lParam)
        {
        // main window handler, called first:
                                                    :
            DoCommandsFrame(hWindow, wParam, lParam);
                                                    :
        };


    LRESULT DoCommandsFrame(HWND hwnd,WPARAM wParam,LPARAM lParam)
        {
                                                    :
            if(!InitializeSPLibrary(Main.GetParameters()))
                nStartupReport += 1;
                                                    :
        }
```

4. Schritt

Im Hauptprogramm wird als erstes der Konstruktor der TMain-Klasse aufgerufen. Dann wird anhand der Variable bNoConfigurationFile festgestellt, ob die Initialisierungsdatei vorhanden ist. Das Programm bricht mit dem Fehlercode 4 die Verarbeitung ab, falls die Variable bNoConfigurationFile den Wert TRUE hat, d.h. die Initialisierungsdatei nicht vorhanden ist. Die Zuweisung des Initialisierungsdateinamens erfolgt dann durch die Methode SetParameters(). Damit sind alle für die Ermittlung des Initialisierungsdateinamens notwendigen Initialisierungen erfolgt.

```
    int PASCAL WinMain( HINSTANCE hInstance,HINSTANCE hPrevInst,LPSTR,int cmdShow)
```

```
{
                                                    :
TMain::hInstance                    = hInstance;
                                                    :
if(bNoConfigurationFile)
        {
        #ifdef GermanVersion
        MessageBox(GetFocus(),"Kein Konfigurationsfile !","Fehler",MBSTOP);
        #else
        MessageBox(GetFocus(),"Configuration file not found !","Failure",MBSTOP);
        #endif
        exit(4);
        }
                                                    :
        wndClass.lpfnWndProc    = FrameWndProc;
                                                    :
// initialize hardware
Main.SetParameters();
                                                    :
};
```

3 Änderungen in den Quelldateien

3.1 Zerlegung der Initialisierungsdatei

Die Initialisierungsdatei wurde in zwei Dateien zerlegt. Die Datei develop.ini enthält die anwenderspezifischen Sektionen, die Datei hardware.ini enthält die arbeitsplatzspezifischen Sektionen.

Die Initialisierungsdateien enthalten folgende Sektionen:

develop.ini

- [Steuerprogramm]
- [ManualMoves]
- [Counter]
- [AreaScan]
- [AreaScanCCD]
- [Scan]
- [Topographie]

hardware.ini

- [GBIB0]
- [Motorx] x=0..n
- [Devicex] x=0..n
- [MOTORSIM]

3.2 Zugriff auf die Initialisierungsdatei

Der Initialisierungsdateiname wurde bisher durch die parameterlose Methode GetCFile() übergeben. Aus diesem Grund habe ich mich dazu entschlossen, eine weitere parameterlose Methode einzuführen, die anstelle des benutzerspezifischen Initialisierungsdateinamens den arbeitsplatzspezifischen Initialisierungsdateinamen zurückliefert. Diese Methode heißt GetHWFile().

Folgende Änderungen wurden durchgeführt:

- Die Methode GetHWFile() wurde in den Dateien l_layer.cpp und l_layer.h implementiert

```
LPCSTR _export WINAPI GetHWFile(void)
  {
  return (LPCSTR)MainP.HWIniFile;
  };
```

- In die Struktur TMainParameters wurde eine weitere Variable HWIniFile aufgenommen und die Klasse TMain um eine zusätzliche Variable HFile erweitert. Diese Änderungen wurden in der Datei comclass.h implementiert.

```
typedef struct
{
  HWND      hWndFrame;
  HWND      hWndClient;
  HINSTANCE hInstance;
  UINT      wm_DrawStatus;
  BOOL      bCreateIniDefaults;
  char      IniFile[MAXPATH];
```

```
  char      HWIniFile[MAXPATH];
} TMainParameters;

class TMain
{
 public:
                                                    :
  char          IFile[MAXPATH];
  char          HFile[MAXPATH];
                                                    :
  TMainParameters  Parameters;
                                                    :
};
```

- In der Datei m_main.cpp wurde eine zusätzliche Variable bNoHardwareFile deklariert, und in den TMain-Konstruktor ein Test eingefügt der ermittelt, ob die Hardwareinitialisierungsdatei vorhanden ist.

```
BOOL        bNoConfigurationFile;
BOOL        bNoHardwareFile;

TMain::TMain(void)
    {
                                                    :
    sprintf(IFile,"%s%s.ini",Main.szDirectory,Name);
    sprintf(HFile,"%shardware.ini",Main.szDirectory);
    if(-1 == OpenFile(IFile,&of,OF_EXIST))
            bNoConfigurationFile = TRUE;
            else
            bNoConfigurationFile = FALSE;
    if((-1 == OpenFile(HFile,&of,OF_EXIST)))
             bNoHardwareFile = TRUE;
            else {
            bNoHardwareFile = FALSE;
                                                    :
    };
```

- In der Methode SetParameters() wird dann die Variable Parameters.HWIniFile mit der Variablen HFile initialisiert.

```
void TMain::SetParameters(void)
    {
                                                    :
    strcpy(Parameters.IniFile,IFile);
    strcpy(Parameters.HWIniFile,HFile);
    };
```

- Wenn die Hardwareinitialisierungsdatei nicht vorhanden ist, wird eine Fehlermeldung ausgegeben und das Programm beendet.

```
int PASCAL WinMain( HINSTANCE hInstance,HINSTANCE hPrevInst,LPSTR,int cmdShow)
  {
                                                    :
  TMain::hInstance                  = hInstance;
                                                    :
  if(bNoConfigurationFile)
        {
        #ifdef GermanVersion
        MessageBox(GetFocus(),"Kein Konfigurationsfile !","Fehler",MBSTOP);
        #else
        MessageBox(GetFocus(),"Configuration file not found !","Failure",MBSTOP);
        #endif
        exit(4);
        }
  if(bNoHardwareFile)
        {
        #ifdef GermanVersion
        MessageBox(GetFocus(),"Kein Hardwarekonfigurationsfile !","Fehler",MBSTOP);
        #else
        MessageBox(GetFocus(),"Hardware Configuration file not found !","Failure",
                    MBSTOP);
        #endif
        exit(4);
        }

                                                    :
  // initialize hardware
  Main.SetParameters();
                                                    :
  };
```

- In den Dateien am9513.cpp, braunpsd.cpp, counters.cpp, motors.cpp und m_arscan.cpp wurde in allen Get/WritePrivateProfile-Aufrufen der Parameter GetCFile() durch den Parameter GetHWFile() ersetzt, da dort ausschließlich auf die arbeitsplatzspezifischen Sektionen zugegriffen wird.

```
WritePrivateProfileString( lpAppName, lpKeyName, lpString, GetHWFile() );
GetPrivateProfileString(lpAppName, lpKeyName, lpDefault, lpReturnedString, nSize,
                        GetHWFile() );
i = GetPrivateProfileInt( lpAppName, lpKeyName, nDefault, GetHWFile()  );
```

3.3 Zugriffsschutz für die Hardwareinitialisierungsdatei

Die Hardwareinitialisierungsdatei wird durch die Verwendung der Attribute „Versteckt“ und „ReadOnly“ vor unbeabsichtigten und beabsichtigten Veränderungen besonders geschützt.

Der Zugriffsschutz über die Dateiattribute wurde in der Datei m_main.cpp implementiert und ist abhängig davon, ob das Programm in 16Bit oder 32Bit kompiliert wird.

- Um auf die Dateiattribute zugreifen zu können musste eine zusätzliche Bibliothek eingebunden werden.

```
#ifndef __WIN32__
#include    <io.h>
#else
#include    <winbase.h>
#endif
```

- Im Konstruktor der Klasse TMain werden der Hardwareinitialisierungsdatei, falls diese existiert, die Attribute „Versteckt“ und „Archiv“ zugewiesen. Dadurch wird das Attribut „ReadOnly“ entfernt.

```
TMain::TMain(void)
   {
                                                    :
      if((-1 == OpenFile(HFile,&of,OF_EXIST)))
         bNoHardwareFile = TRUE;
      else {
         bNoHardwareFile = FALSE;
#ifndef __WIN32__
          _rtl_chmod(HFile,1,FA_HIDDEN + FA_ARCH);
#else
          SetFileAttributes(HFile,FILE_ATTRIBUTE_HIDDEN + FILE_ATTRIBUTE_ARCHIVE);
#endif
      }
                                                    :
   }
```

- Im Destruktor der Klasse TMain werden der Hardwareinitialisierungsdatei die „Attribute“ „Versteckt“, „Archiv“ und „ReadOnly“ zugewiesen.

```
TMain::~TMain(void)
   {
                                                    :
#ifndef __WIN32__
  _rtl_chmod(HFile,1,FA_HIDDEN + FA_ARCH + FA_RDONLY);
#else
  SetFileAttributes(HFile,FILE_ATTRIBUTE_HIDDEN + FILE_ATTRIBUTE_ARCHIVE \
                    + FILE_ATTRIBUTE_READONLY);
#endif
   };
```

Insgesamt wurden ca. 197 Quelltextzeilen in 9 Dateien geändert.

4 Ausblick auf weitere Programmverbesserungen

Durch die Teilung der Initialisierungsdatei und den besonderen Schutz der Hardwareinitialisierungsdatei ist es jetzt möglich, über weitere Verbesserungen des XCTL-Programms nachzudenken. Das Programm kann nun so verändert werden, das es möglich wird, personalisierte Initialisierungsdateien zu verwenden. Damit hat der Benutzer die Möglichkeit, das Programm immer mit den gewohnten Einstellungen zu starten. Dazu zählen u.a. Fenstergrößen und -positionen, Namen von Experimentator und Arbeitsgruppe müssen nicht bei jedem Programmstart neu eingegeben werden, und der Einsatz einer Initialisierungsdatei für verschiedene Messplätze ist möglich. Somit kann der Arbeitsablauf für die einzelnen Mitarbeiter deutlich verbessert werden.