Technical Article

DriveLock – Custom file type detection

Technical Documentation

DriveLock SE 2022

Table of Contents

Introduction

DriveLock features access control to certain files types based on the content of these files. Using file filtering an administrator can configure which file types are allowed to be read and / or written to removable storage. As in Windows file types are basically determined by a file extension any user or malicious software can simply rename files to another file extension, for example a user could rename its music collection from *.MP3 to *.DOC so that a file filter “thinks” it must be a collection of probably allowed Word documents.

To circumvent this type of attack, DriveLock’s file filter scans the content of each file to ensure the file content is from the type the file extension implies. DriveLock has built-in content scanning and file type recognition for the most commonly used file types:

Application	File extensions
Executable files	386 AX COM CPL DLL EXE FLT OCX SCR SYS VXD
Microsoft Access	ACCDB ACCDE ACCDT ACCDR (Access 2007 or newer) DB MDB MDE SNP (Access) DOC DOT WIZ (Word 2007 or newer) DOCX DOCM DOTX DOTM (Word) MPP (Project) ONE (OneNote) PPS PPT PPZ (PowerPoint) PST (Outlook) PPTX PPTM POTX POTM (PowerPoint 2007 or newer) PPSM PPSX (PowerPoint 2007 or newer) VSD (Visio) WPS (Works) XLS XLA XLT (Excel) XLSX XLSM XLSB XLTX XLAM (Excel 2007)
Microsoft Installer	MSI MSP MSM
Compressed archives	ACE ARC ARJ CAB GZ RAR TGZ ZIP
Multimedia	AVI (AVI Video) BUP IFO VOB (DVD) ITL (iTunes Library) M4P M4A M4V (Apple Video / Music) MID MIDI (MIDI) MOV (Quicktime) MP2 MP3 MPG MPEG (MPEG) WAV (Windows WAV audio) WMA WMV (Windows Media)

Application	File extensions	
Windows system files	ANI ICO	(Cursor, Icon)
	CDR	(CD Audio)
	CHM	(Windows Help)
	MSC	(Management Console)
	TTF	(TrueType Font)
Pictures	BMP	(Bitmap)
	GIF	(GIF)
	JFIF JPE JPEG JPG	(JPEG)
	MDI	(MS Document Imaging)
	PNG	(PNG)
	PSPIMAGE	(PaintShop Pro)
	TIF TIFF	(TIFF)
	WMF	(Windows Metafile)
Backup files	BKF	
Encrypted containers	DLV	
Technical drawings	DWG	(AutoCAD)
CD/DVD images	ISO	
Other executable files	JAR	
Other documents	PDF	(Adobe PDF)
	PS	(Postscript)
	RTF	(Rich Text Format)
	SWF	(Shockwave Flash)
Virtual hard disks	VMDK	(VMWare)
	VHD	(Windows)

Other file types can be detected using "*File type definitions*" function under "*Drives"* in the DriveLock Management Console, which is described in detail in this document.

Custom file type definitions

DriveLock contains built-in file type definitions for the most commonly used file types. These definitions are pre-configured within each DriveLock Agent, so it is not necessary to configure anything in order to detect these types of files.

The configuration node "*File type definitions*" under "*Drives*" in the DriveLock Management Console contains all file type definitions in addition to the built-in definitions. This means in a default configuration this list is empty and DriveLock uses the built-in definitions only.

Overriding built-in definitions

If a built-in definition does not suit the needs of a certain configuration, it is possible to override or extend the built-in definition. This is done by defining a custom file type definition for a built-in file type. Custom definitions always take precedence over the built-in definitions.

To get an overview of how the built-in definitions work and if they suit your needs, right-click on **File type definitions** then choose **Create built-in definitions** from the *All Tasks* menu:

This creates custom file type definitions for each of the built-in file types. Working with these definitions is the same as when creating custom file type definitions for non-built-in types.

Creating custom file type definitions

If it is necessary to detect file types not pre-configured within DriveLock, a custom file type definition needs to be created.

Prior to this, correct information is needed how a certain file type can be detected. For performance reasons DriveLock only reads the first 10 KB of a file to be able to check the contents of the file's header. So detection is based on the file's header. In general most file types contain some kind of signature or header which is used within the corresponding application to detect if the file is valid. File format documentation can often be found on the application manufacturer's web site.

The following sample walk through show's how to create a new custom file definition for the SLN (Microsoft Visual Studio Solution) file format:

The file format specification shows that there are two possible headers of a SLN file:

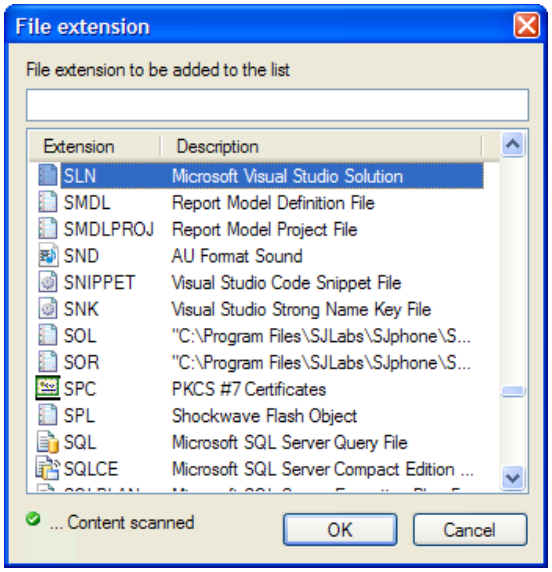
```
000000  4D 69 63 72 6F 73 6F 66  74 20 56 69 73 75 61 6C  Microsoft Visual
000010  20 53 74 75 64 69 6F 20  53 6F 6C 75 74 69 6F 6E   Studio Solution
000020  20 46 69 6C 65 2C 20 46  6F 72 6D 61 74 20 56 65   File, Format Ve
000030  72 73 69 6F 6E 20 39 2E  30 30 0D 0A 23 20 56 69  rsion 9.00..# Vi
000040  73 75 61 6C 20 53 74 75  64 69 6F 20 32 30 30 35  sual Studio 2005
```

```
000000  EF BB BF 0D 0A 4D 69 63  72 6F 73 6F 66 74 20 56  .....Microsoft V
000010  69 73 75 61 6C 20 53 74  75 64 69 6F 20 53 6F 6C  isual Studio Sol
000020  75 74 69 6F 6E 20 46 69  6C 65 2C 20 46 6F 72 6D  ution File, Form
000030  61 74 20 56 65 72 73 69  6F 6E 20 39 2E 30 30 0D  at Version 9.00.
```

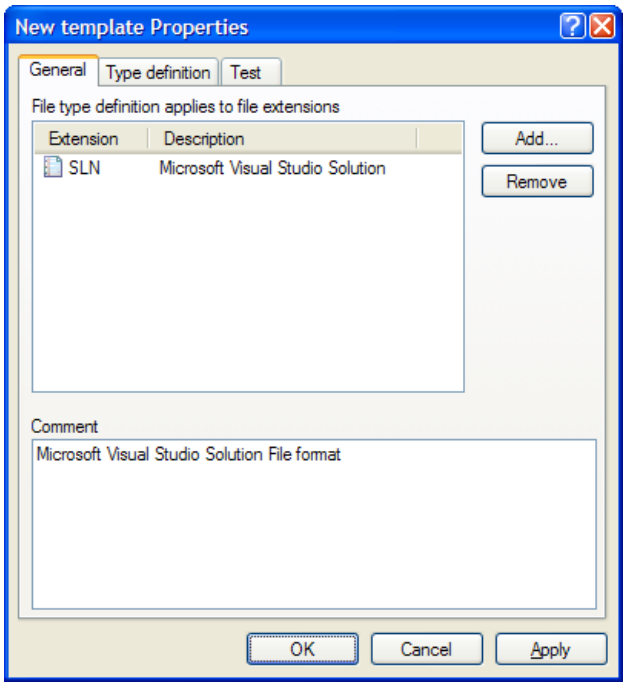
The file is basically a text file starting with "Microsoft Visual Studio Solution File" but it may be encoded as ASCII (first file) or as UTF-8 (second file). In the UTF-8 case the file starts with a Unicode BOM (Byte Order Mark) followed by a line feed.

In DriveLock create a new file type definition (Right-click on **File type definitions** then choose "*New → File type definition…*"):

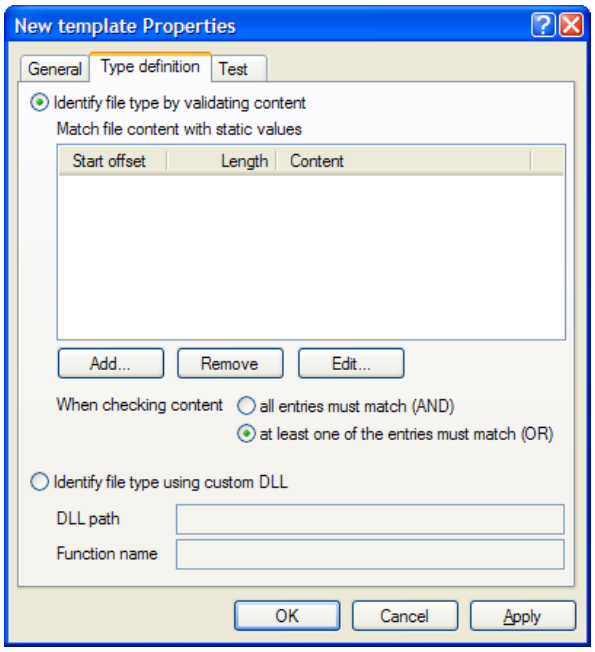
An empty file type definition property sheet appears. Click **Add…** to add a file extension to the list of file extensions valid for this definition:



Choose SLN from the list (or type SLN in the edit box). Then click OK.



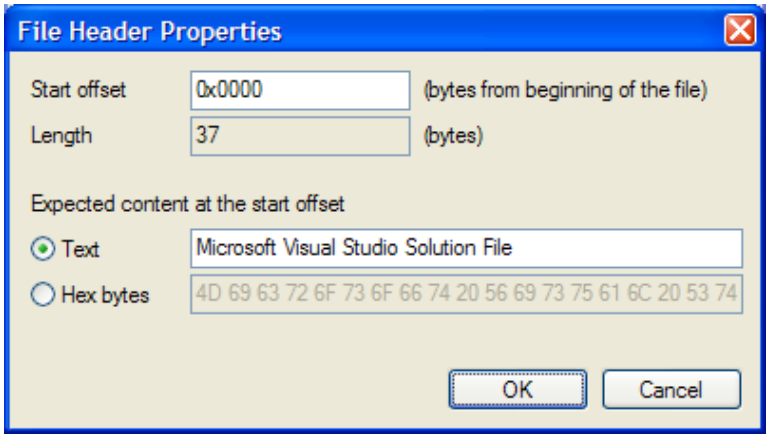
Select the *Type definition* tab to switch to the definition page:



For the SLN file format we can use the "*Identify file type by validating content*" option. If a more complex file type needs to be validated, you can create a custom DLL which scans the file content. This is described later in this document.

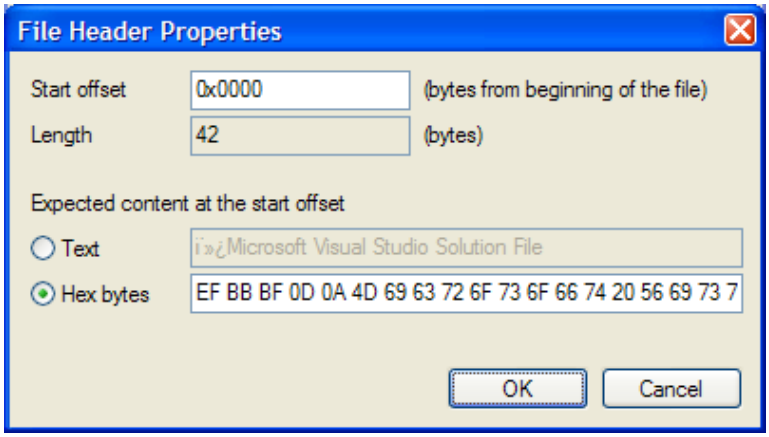
For SLN files, two rules need to be created, one for each of the two possible headers in a SLN file. As only one of the headers is present in the file, you must select "*…at least one of the entries must match (OR)*".

Click Add… to create the first header validation rule:



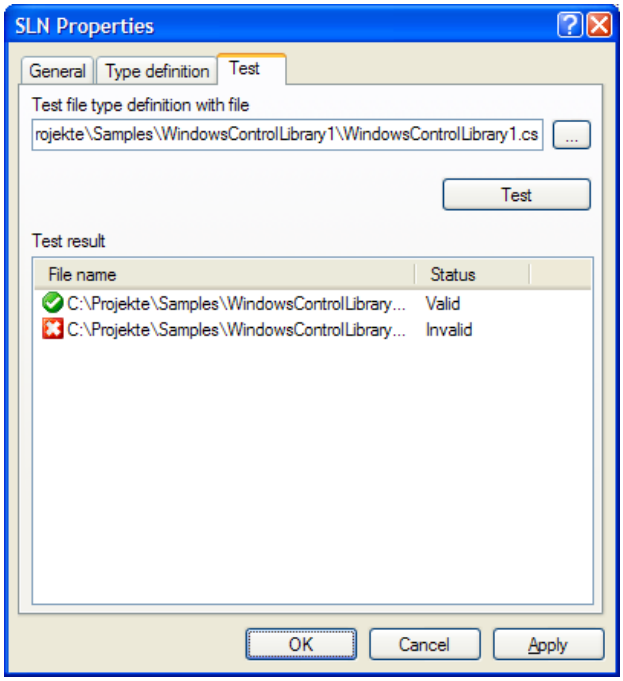
We want to check the file from the beginning, so "*Start offset*" is "0" As we want to check a plain text, we choose "*Text*" and then enter the text to check for: "`Microsoft Visual Studio Solution File`". Then click OK to save this rule.

Click Add… again to create the second header validation rule:



Again we want to check from the beginning of the file, so "*Start offset*" is 0. In UTF-8 format, we cannot check for a plain text, we choose "*Hex bytes*". Then enter the hex bytes as in the file format specification. Click OK to save this rule.

To test the definition click on the "*Test*" tab:



Select some files (SLN and other formats) by clicking "**…**", then click **Test** to apply the type definition to the file. Verify that the selected SLN files are valid while other file types are invalid.

Custom DLL development

As described earlier a file type definition can be created by using simple validation rules that check a header of a file. While this is suitable for most file types, some types need more complex checks to determine their file type correctly.

This can be accomplished by developing a custom DLL.

As an example we create a file type definition for a fictive GUID file type, which is a text file that starts with a valid GUID:

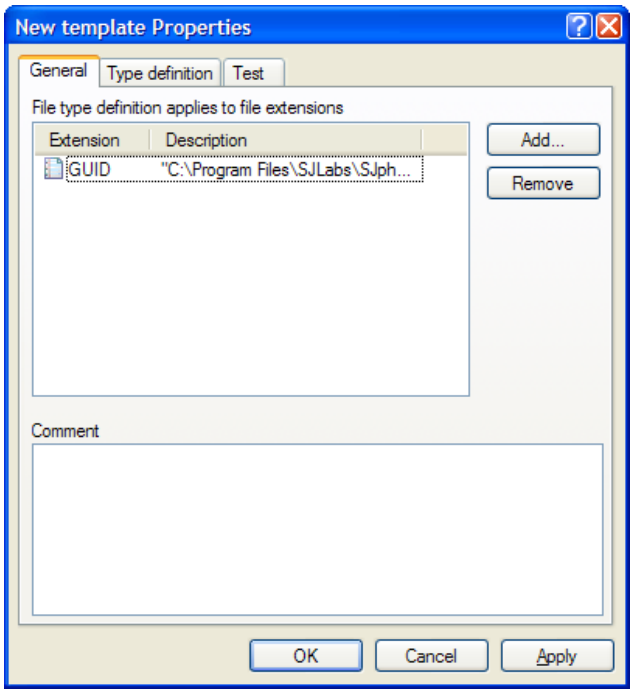
```
000000  66 64 64 37 33 61 61 64  2D 66 33 30 35 2D 34 33  fdd73aad-f305-43
000010  37 39 2D 39 62 39 39 2D  66 38 30 32 32 39 32 64  79-9b99-f802292d
000020  32 64 66 61 0D 0A 43 6F  6E 74 65 6E 74 0D 0A 43  2dfa..Content..C
000030  6F 6E 74 65 6E 74 0D 0A  43 6F 6E 74 65 6E 74 0D  ontent..Content.
000040  0A 43 6F 6E 74 65 6E 74  0D 0A 43 6F 6E 74 65 6E  .Content..Conten
000050  74 0D 0A 43 6F 6E 74 65  6E 74 0D 0A              t..Content..
```

Checking for a valid GUID cannot be done using static value rules. Therefore we need to create a custom DLL performing this check.

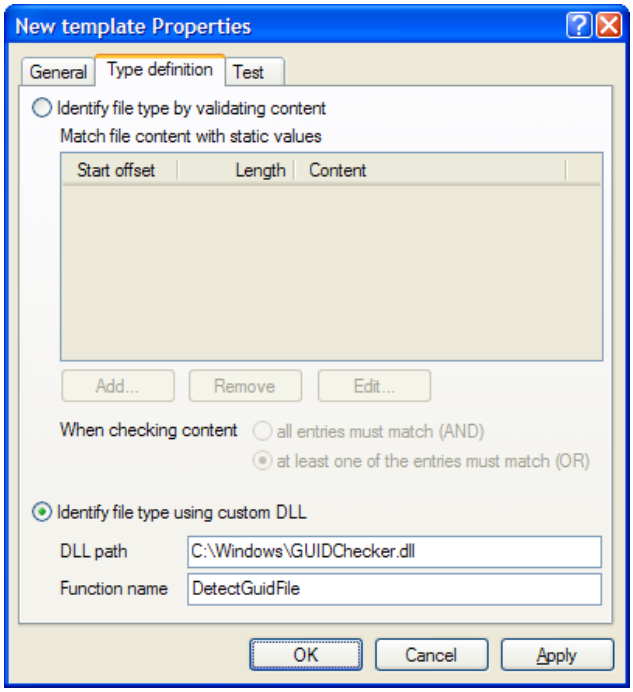
Configuring a custom DLL

In DriveLock create a new file type definition (Right-click on "*File type definitions*" then select "*New → File type definition…*"):

An empty file type definition property sheet appears. Click Add… to add a file extension to the list of file extensions valid for this definition. Type `GUID` in the Edit box, then click OK.



Click on "*Type definition*" to switch to the definition page:



Choose "*Identify file type using custom DLL*". Enter the full path to the DLL in "***DLL path***". This path must be the same on all workstations where DriveLock Agent uses this file type definition. The custom DLL (in our case *GUIDChecker.dll*, located under "*C:\Windows*") must be copied to the workstations using some external mechanism. DriveLock does not copy this file and simply expects the file located under the configured path.

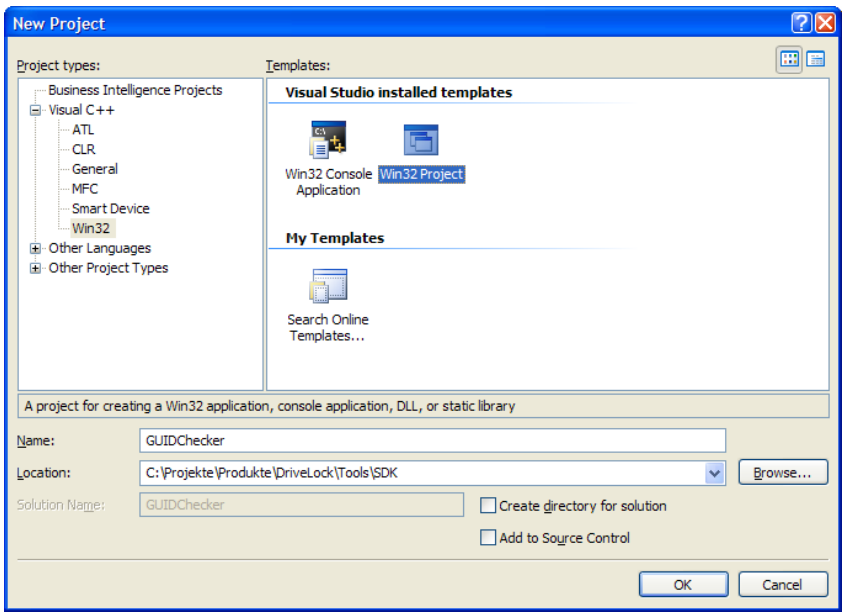
Enter the function name exported from the DLL in "*Function name*" (see later how to create the function).

You can use the "*Test*" tab to verify your configuration works correctly.

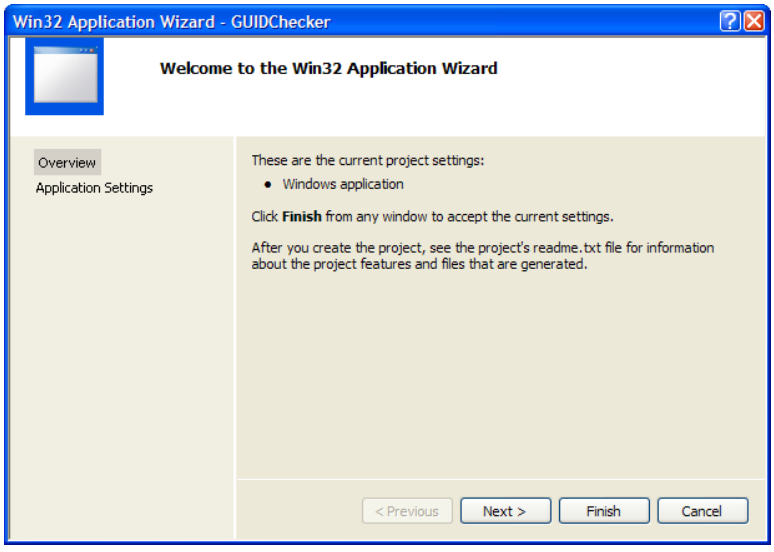
Creating a custom DLL

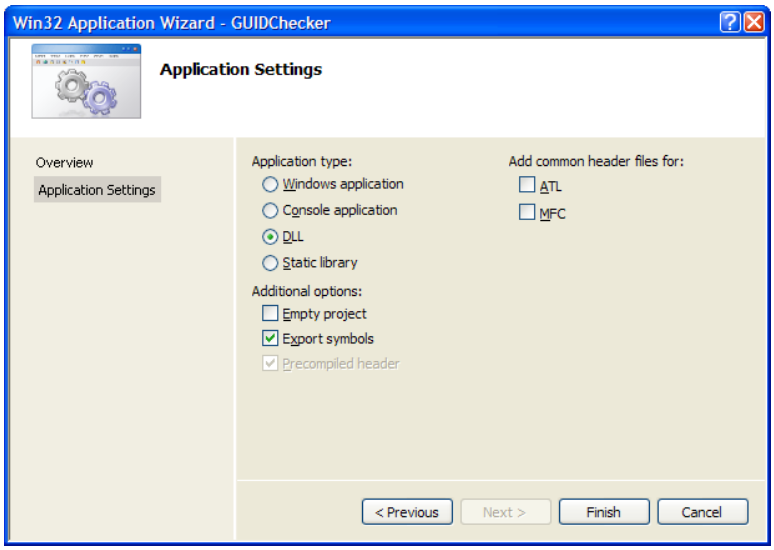
The custom DLL itself needs to be developed in an appropriate programming language. We recommend using Microsoft Visual Studio to create the DLL. The following documentation shows the steps to create a DLL checking GUID files.

Start Microsoft Visual Studio. Create a new Project (*File* → *New* → *Project…*). Create a Win32 Project:



Walk through the "Windows Application" wizard, creating a DLL that exports some symbols:





After the project is created by the wizard, it contains a C++ source and header file with some sample exports.

A custom file type detection function is a C function defined as:

```
BOOL DetectProc(LPBYTE content, DWORD cbSize)
```

The function takes two parameters:
content is a pointer to a buffer containing the first 10 KB of the file. Do not attempt to change the contents of this buffer as unpredictable results may occur.
cbSize contains the actual size of the buffer (if the file is smaller than 10 KB).

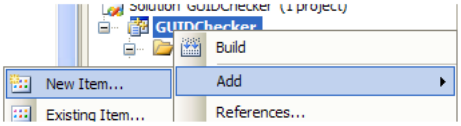
The function returns *TRUE* if the file content meets the specification, thus the file is of the detected type, *FALSE* otherwise.

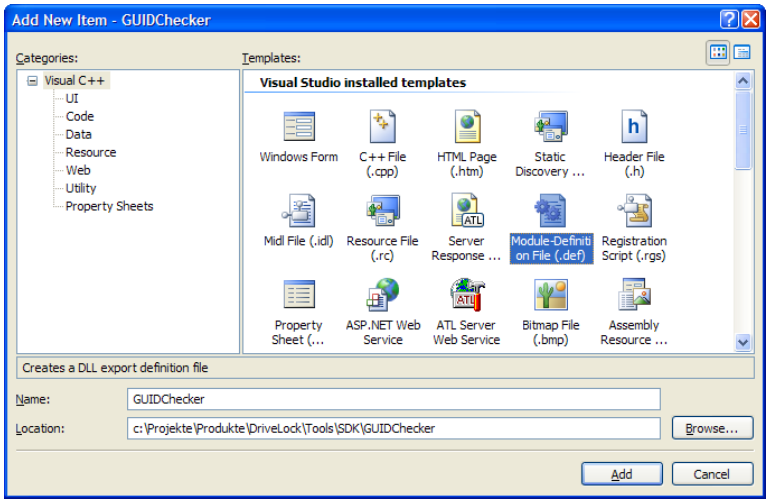
The function to detect a GUID file is implemented as follows:

```
GUIDCHECKER_API BOOL DetectGuidFile(LPBYTE content, DWORD cbSize)
{
  unsigned char szUuid[GUID_LENGTH + 1];
  UUID binUuid;

  // If the file is less than a GUID long, it must be invalid
  if (cbSize < GUID_LENGTH)
    return FALSE;

  // Copy the GUID to our internal buffer
  ZeroMemory(szUuid, sizeof(szUuid));
  memcpy(szUuid, content, GUID_LENGTH);

  // Try to convert the text GUID to a binary GUID, if this succeeds
  // our file must be valid
  if (UuidFromStringA(szUuid, &binUuid) == RPC_S_OK)
    return TRUE;

  return FALSE;
}
```

Please note that when compiling the DLL as created by the Visual Studio wizard, the *DetectGuidFile* function is exported with C++ name mangling which is not suitable for DriveLock.

To create a plain C-style export, add a Module Definition File to the project:





The file GUIDChecker.def has the following content:

```
LIBRARY     "GUIDChecker"

EXPORTS     DetectGuidFile
```