GBS ELEKTRONIK GmbH

Quick IDENTIFY

(Version 1.06)

User's Manual

(for developers)

All trademarks of other companies used within this document are only used for identification. They are property of the according manufacturers and are acknowledged herewith.

Author André Birnbaum

Document number 04310-001I01a

Revision 2017-11-10

Address:

GBS Elektronik GmbH
Bautzner Landstrasse 22
01454 Grosserkmannsdorf

Tel.: + 49 - (0) - 351 - 21 70 07 - 0
Fax.: + 49 - (0) - 351 - 21 70 07 - 21
E-Mail: kontakt@gbs-elektronik.de
Homepage: http://www.gbs-elektronik.de/

Table of Contents

0 Introduction

Quick IDENTIFY is a function library intended to be linked to third party programs which want to do nuclide identification. Quick IDENTIFY[1] is equivalent to the QuickID function of the IDENTIFY program for the desktop PC. About the working method of IDENTIFY read the user's manual of the program.

For Microsoft Windows 95/98/NT/2000/ME/XP/Vista/7/8/10 and Microsoft Windows CE[2], dynamic link libraries (identify.dll, identif2.dll[3]) are available. For UNIX operating systems, the source code is supplied. In this way it is possible to compile and link the IDENTIFY software to own programs on different UNIX operating systems. More about UNIX and Quick IDENTIFY see chapter 4.

Some functions use the Microsoft Foundation Class `CString`. These functions are only available on operating systems that provide the corresponding library.

The user of Quick IDENTIFY needs the IDENTIFY program because he has to prepare the setup (identify.set) and the nuclide library (standard.lib) with it. In order to run Quick IDENTIFY four files are necessary: identify.dll (MS Windows only), r5.dat, standard.lib (or another nuclide library) and identify.set (or another setup). R5.dat or r6.dat, standard.lib and identify.set must be in the same directory as the library or UNIX program. All three files are delivered together with the IDENTIFY program or they are created or edited by it.

Of course, if you want to be independent of the IDENTIFY program, you can also develop your own program to create or edit the library or setup. About the items of the setup file that are considered by Quick IDENTIFY see chapter 2.

1 Functions and structures

1.1 LoadSetup

The function loads a setup.

```
int IDENTIFY_API LoadSetup(LPCTSTR lpszSetupFile);
```

Parameter

lpszSetupFile	Pointer to a null-terminated string containing the path and file name of the setup.

Return value

If the function succeeds the return value is ERROR_NONE. The possible return values are ...

```
#define ERROR_NONE                  0
#define ERROR_MEMORY                1
#define ERROR_FILE_SYNTAX           2
#define ERROR_FILE_OPEN             3
```

Remarks

If you do not load a setup file explicitly with this function, the identify.set will be automatically loaded by the functions that need the setup.

[1] See „Revision history", chapter 5

[2] Pocket IDENTIFY

[3] If MFC is not supported on your device, you can use identif2.dll instead of identify.dll. It offers the same functions without using MFC.

1.2 LoadNuclideLibrary

The function loads a nuclide library.

```
int IDENTIFY_API LoadNuclideLibrary(LPCTSTR lpszLibraryFile);
```

Parameter

lpszLibraryFile	Pointer to a null-terminated string containing the path and file name of the nuclide library.

Return value

If the function succeeds the return value is ERROR_NONE. For the possible return values see above.

Remarks

If you do not load a nuclide library file explicitly with this function, a nuclide library *(default: standard.lib)* will be automatically loaded by the functions that need nuclides from a library.

1.3 LoadNuclideProperties

The function loads additional nuclide properties from a nuclide properties file *(*.npf)*. The properties are used within the function *RunIdentifyFlag*.

```
int IDENTIFY_API LoadNuclideProperties(LPCTSTR lpszPropertiesFile);
```

Parameter

lpszPropertiesFile	Pointer to a null-terminated string containing the path and file name of the nuclide properties file.

Return value

If the function succeeds, the return value is ERROR_NONE. The following value is only returned by LoadNuclideProperties.

```
#define ERROR_NO_NUCLIDES_LOADED     4
```

For further possible return values see above.

Remarks

If you do not load a nuclide properties file explicitly with this function, a nuclide properties file *(default: [name of the nuclide library].npf)* will be automatically loaded by the functions that need additional nuclide properties.

Before additional nuclide properties can be loaded, the nuclides **must** be loaded with LoadNuclideLibrary.

Nuclide properties files are ASCII files with the following sysntax:

Nuclide name: [MEDICAL] [NATURAL] [INDUSTRIAL] [NUCLEAR] [SUSPECT]

```
e.g.   K  40: NATURAL
       Ga 67: MEDICAL
```

1.4 RunIdentify ...

There are six different functions that all do nuclide identification. In principle all functions do the same. The reason for this variety is to offer functions for different programming languages and different uses. RunIdentify and RunIdentifyEx do exactly the same like QuickID of the IDENTIFY program. RunIdentify is destinated for use with and RunIdentifyEx for use without MFC.

RunIdentifyFlag is destinated for automatic evaluation. The program which calls this function has the possibility to evaluate the flags of the return value.

RunIdentifyWithBackground, RunIdentifyExWithBackground and RunIdentifyFlagWithBackground include a background spectrum in the calculation.

```
CString IDENTIFY_API RunIdentify
                     (unsigned long  *lpulSpectrum,
                      short           nChannels,
                      float           fLiveTime,
                      float           fRealTime,
                      ECAL_IDENTIFY  *lpEcal=NULL);


void    IDENTIFY_API RunIdentifyEx
                     (unsigned long  *lpulSpectrum,
                      short           nChannels,
                      float           fLiveTime,
                      float           fRealTime,
                      LPTSTR          lpszResult,
                      unsigned int    unLength,
                      ECAL_IDENTIFY  *lpEcal=NULL);


DWORD   IDENTIFY_API RunIdentifyFlag
                     (unsigned long  *lpulSpectrum,
                      short           nChannels,
                      float           fLiveTime,
                      float           fRealTime,
                      LPTSTR          lpszResult=NULL,
                      unsigned        int unLength=0,
                      unsigned long   ulTextFlags=IDF_WRITE_NOTHING,
                      ECAL_IDENTIFY  *lpEcal=NULL);


CString IDENTIFY_API RunIdentifyWithBackground
                     (unsigned long  *lpulSpectrum,
                      unsigned long  *lpulSpectrumBkgnd,
                      short           nChannels,
                      float           fLiveTime,
                      float           fRealTime,
                      float           fLiveTimeBkgnd,
                      ECAL_IDENTIFY  *lpEcal=NULL);


void    IDENTIFY_API RunIdentifyExWithBackground
                     (unsigned long  *lpulSpectrum,
                      unsigned long  *lpulSpectrumBkgnd,
                      short           nChannels,
                      float           fLiveTime,
                      float           fRealTime,
                      float           fLiveTimeBkgnd,
                      LPTSTR          lpszResult,
                      unsigned int    unLength,
                      ECAL_IDENTIFY   *lpEcal=NULL);


DWORD   IDENTIFY_API RunIdentifyFlagWithBackground
                     (unsigned long  *lpulSpectrum,
                      unsigned long  *lpulSpectrumBkgnd,
                      short           nChannels,
                      float           fLiveTime,
                      float           fRealTime,
                      float           fLiveTimeBkgnd,
                      LPTSTR          lpszResult=NULL,
                      unsigned int    unLength=0,
                      unsigned long   ulTextFlags=IDF_WRITE_NOTHING,
                      ECAL_IDENTIFY  *lpEcal=NULL);
```

Parameters

lpulSpectrum	Pointer to an array that contains the spectrum.
lpulSpectrumBkgnd	Pointer to an array that contains the background spectrum.
nChannels	Channel number[4] of the spectrum. It is equal the size of the arrays at which *lpulSpectrum* and *lpulSpectrumBkgnd* point.
fLiveTime	The live time of the measurement.
fRealTime	The real time of the measurement.
fLiveTimeBkgnd	The live time of the background measurement.
lpszResult	Pointer to a null-terminated string. The string contains the result of the nuclide identification. At RunIdentifyFlag and RunIdentifyFlagWithBackground: If this parameter is unused, set it to NULL.
unLength	Maximum length of the result string. At RunIdentifyFlag and RunIdentifyFlagWithBackground: If *lpszResult* is NULL, set it to 0.
ulTextFlags	Determines the text written to *lpszResult* at RunIdentifyFlag and RunIdentifyFlagWithBackground.

```
#define IDF_WRITE_NOTHING             0x00000001L
#define IDF_WRITE_HEADER              0x00000002L
#define IDF_WRITE_ERRORS              0x00000004L
#define IDF_WRITE_ECAL                0x00000008L
#define IDF_WRITE_RESULT              0x00000010L
#define IDF_WRITE_PEAKS               0x00000020L[5]
#define IDF_WRITE_PEAKS_NOT_ASSIGNED  0x00000020L
#define IDF_WRITE_UNSURE              0x00000040L
#define IDF_WRITE_MDA                 0x00000080L
#define IDF_WRITE_RELATION            0x00000100L
#define IDF_WRITE_PEAKS_FOUND         0x00000200L
#define IDF_WRITE_LOCAL_DOSE_RATE     0x00000400L
#define IDF_WRITE_ID_QUALITY          0x00000800L[6]
#define IDF_WRITE_ALL                 0xFFFFFFFEL
```

lpEcal	Pointer to an ECAL_IDENTIFY structure. The parameter is optional. If *lpEcal* is NULL, the existing energy calibration is used. For more about the ECAL_IDENTIFY structure see section 1.7.

[4] There is a difference between the results, if different channel numbers are passed although the spectrum data are identical.

[5] Up to version 1.03, only the peaks not assigned are written to the result string. Since version 1.04, all peaks found are written to the result string. Therefore, the flags IDF_WRITE_PEAKS_NOT_ASSIGNED and IDF_WRITE_PEAKS_FOUND were added. The new flag IDF_WRITE_PEAKS_NOT_ASSIGNED is equal the old flag IDF_WRITE_PEAKS.

[6] New since version 1.05.0003

Return value

RunIdentify,
RunIdentifyWithBackground: A string that contains the result of the nuclide identification.

RunIdentifyEx,
RunIdentifyExWithBackground: No return value. The result of the nuclide identification is written to *lpszResult*.

RunIdentifyFlag,
RunIdentifyFlagWithBackground: Flags that contain the result of the nuclide identification. The result of the nuclide identification is besides written to *lpszResult*.

```
#define RESULT_MEMORY_ERROR                        0x00000001L
#define RESULT_NO_SETUP                            0x00000002L
#define RESULT_NO_NUCLIDES_LOADED                  0x00000004L
#define RESULT_NO_PROPERTIES_LOADED                0x00000008L
#define RESULT_UNCLASSIFIED_NUCLIDES               0x00000010L
#define RESULT_UNCLASSIFIED_NUCLIDES_UNSURE        0x00000020L
#define RESULT_UNCLASSIFIED_NUCLIDES_VERY_UNSURE   0x00000040L
#define RESULT_MEDICAL_NUCLIDES                    0x00000080L
#define RESULT_MEDICAL_NUCLIDES_UNSURE             0x00000100L
#define RESULT_MEDICAL_NUCLIDES_VERY_UNSURE        0x00000200L
#define RESULT_NATURAL_NUCLIDES                    0x00000400L
#define RESULT_NATURAL_NUCLIDES_UNSURE             0x00000800L
#define RESULT_NATURAL_NUCLIDES_VERY_UNSURE        0x00001000L
#define RESULT_NUCLEAR_NUCLIDES                    0x00002000L
#define RESULT_NUCLEAR_NUCLIDES_UNSURE             0x00004000L
#define RESULT_NUCLEAR_NUCLIDES_VERY_UNSURE        0x00008000L
#define RESULT_INDUSTRIAL_NUCLIDES                 0x00010000L
#define RESULT_INDUSTRIAL_NUCLIDES_UNSURE          0x00020000L
#define RESULT_INDUSTRIAL_NUCLIDES_VERY_UNSURE     0x00040000L
#define RESULT_SUSPECT_NUCLIDES                    0x00080000L
#define RESULT_SUSPECT_NUCLIDES_UNSURE             0x00100000L
#define RESULT_SUSPECT_NUCLIDES_VERY_UNSURE        0x00200000L
#define RESULT_UNASSIGNABLE_PEAKS                  0x00400000L
```

The first four flags should not be returned if enough memory is available and all necessary files have been loaded. The other flags indicate the actual result. If a flag is set, one or several nuclides with the corresponding certainty or unassignable peaks were found.

Remarks

Before one of this functions can be called, a setup and a nuclide library should be loaded. Besides, RunIdentifyFlag needs additional nuclide properties. (See the functions above.) If no setup has been loaded before, identify.set will be loaded by the function. If no nuclide library has been loaded before, the nuclide library file according the setup will be loaded by the function. If this file cannot be found neither in the given path[7] nor in the working directory, the standard.lib will be loaded. If no additional nuclide properties have been loaded before RunIdentify is called, the nuclide properties file corresponding to the name of the nuclide library will be loaded by the function.

If you pass an energy calibration to the function, the passed energy calibration will be used as long as no other energy calibration has been passed, loaded or calculated before.

[7] In case of Pocket IDENTIFY the path in the setup file may be wrong, because the setup file is created on the PC.

1.5 CalibrateEu152, CalibrateEu152Ex

This functions do an automatic energy calibration using an Eu152 spectrum.

```
CString IDENTIFY_API CalibrateEu152   (unsigned long  *lpulSpectrum,
                                       short           nChannels,
                                       ECAL_IDENTIFY  *lpECal=NULL,
                                       BOOL            bSave=TRUE);


void    IDENTIFY_API CalibrateEu152Ex (unsigned long  *lpulSpectrum,
                                       short           nChannels,
                                       LPTSTR          lpszResult,
                                       unsigned int    unLength,
                                       ECAL_IDENTIFY  *lpECal=NULL,
                                       BOOL            bSave=TRUE);
```

Parameters

lpulSpectrum	Pointer to an array that contains the spectrum.
nChannels	Channel number of the spectrum. It is equal the size of the array at which *lpulSpectrum* points.
lpszResult	Pointer to a null-terminated string. The string contains the result of the energy calibration.
unLength	Maximum length of the result string.
lpEcal	Pointer to an ECAL_IDENTIFY structure. The parameter is optional. If *lpEcal* is NULL, the existing energy calibration is used as coarse energy calibration. In this case the setup should be loaded before (see section 1.1, „LoadSetup"). If no setup has been loaded before, identify.set will be loaded. If you pass an ECAL_IDENTIFY structure, either it contains a valid coarse energy calibration or the existing energy calibration is used. In this case the setup should be loaded before. (see section 1.1, „LoadSetup") If no setup has been loaded before, identify.set will be loaded. If the function is successful, the structure contains the new energy calibration. For more about the ECAL_IDENTIFY structure see section 1.7.
bSave	If *bSave* is TRUE, the result will be saved in the setup file. If the LoadSetup function has been run, the according setup file will be changed, otherwise identify.set.

Return value

A string that contains the result of the energy calibration.

Remarks

The energy calibration calculated by CalibrateEu152 or CalibrateEu152Ex will be used as long as no other energy calibration is passed, loaded or calculated. If you pass an energy calibration to CalibrateEu152 or CalibrateEu152Ex and the function fails, the passed energy calibration will be the new one.

1.6 Odl

```
double Odl();
```

Return value

This function returns the local dose rate (*German:* **O**rts**d**osis**l**eistung) of the spectrum that has been analyzed most recently with one of the RunIdentify functions.

Remarks

Note, do not execute CalibrateEu152 or CalibrateEu152Ex between RunIdentify and Odl, because CalibrateEu152 and CalibrateEu152Ex change the spectrum data.

The local dose rate calculation assumes a 40x40mm NAI detector.

1.7 ECAL_IDENTIFY

The `ECAL_IDENTIFY` structure allows to pass an individual energy calibration to each call of RunIdentify, RunIdentifyEx, RunIdentifyFlag, CalibrateEu152 or CalibrateEu152Ex.

IDENTIFY uses the formula: Energy = A * Channel + B + C * Channel ².

```
struct ECAL_IDENTIFY
{ short nPoints;
  short nFlags;
  float fCh1;
  float fE1;
  float fCh2;
  float fE2;
  float fCh3;
  float fE3;
  float fA;
  float fB;
  float fC;
};
```

Members

nPoints Number of points used for the energy calibration. This value can be ...

- -1 *fA*, *fB* and *fC* contain a valid energy calibration which shall be used by the function.
- 0 The existing energy calibration of IDENTIFY shall be used.
- 1 An one-point energy calibration shall be made.
- 2 A two-point energy calibration shall be made.
- 3 A three-point energy calibration shall be made.

nFlags The flags are set by CalibrateEu152 and CalibrateEu152Ex (see section 1.5). Possible flags are:

```
#define ECAL_SUCCESSFUL             0x0001
#define ECAL_FAILED          0x0002
```

fCh1 Channel of the first calibration point. Used if *nPoints* greater than 0.

fE1 Energy of the first calibration point. Used if *nPoints* greater than 0.

fCh2 Channel of the second calibration point. Used if *nPoints* greater than 1.

fE2 Energy of the second calibration point. Used if *nPoints* greater than 1.

fCh3 Channel of the third calibration point. Used if *nPoints* greater than 2.

fE3 Energy of the third calibration point. Used if *nPoints* greater than 2

fA Factor A of the energy calibration formula. Used if *nPoints* equal –1, otherwise it returns calculated value.

fB Factor B of the energy calibration formula. Used if *nPoints* equal –1, otherwise it returns calculated value.

fC Factor C of the energy calibration formula. Used if *nPoints* equal –1, otherwise it returns calculated value.

Remarks

The internal energy calibration procedure of IDENTIFY checks the values for correctness. It is recommend to use energy/channel pairs to give the calibration, in this way also calibration errors can be taken into accont. For more information about energy calibration see the IDENTIFY user's manual.

1.8 InitIdentify

This function is only needed for the UNIX version of Quick IDENTIFY. It must be called before any other function can be called.

```
BOOL InitIdentify(const char *lpszPath);
```

Parameters

lpszPath	Pointer to a null-terminated string. The string contains the default working directory.

Return value

If the function succeeds, the return value is TRUE.

1.9 ExitIdentify

This function is only needed for the UNIX version of Quick IDENTIFY. It must be called before the program exits.

```
void ExitIdentify();
```

2 Setup file (identify.set)

The setup file is made by the IDENTIFY program. Quick IDENTIFY do not read all items. The items read by Quick IDENTIFY are listed in the following table:

Reportoption:	FWHM_FWHM2:
Standard_Bibliothek:	FWHM_EXP2:
Anzeige_Info:[8]	FWHM_Energie2:
Sprache_englisch:	Energieeichung_Steigung:[9]
Sprache_russisch:	Energieeichung_Offset:[9]
Einheit_Bequerel:	Energieeichung_Quadrat:[9]
Peaksuchempfindlichkeit:	Kalib1_Kanal:[9]
Peaksuchaufloesung:[10]	Kalib2_Kanal:[9]
Temperaturschwankungen:	Kalib3_Kanal:[9]
Detektorflaeche:	Kalib1_Energie:[9]
Detektordicke:	Kalib2_Energie:[9]
relative_Efficiency:	Kalib3_Energie:[9]
Detektorfenster_Aluminium:	Detektortyp:
Detektorfenster_Beryllium:	Detektorangaben_Genauigkeit:
Detektorfenster_Eisen:	Absorber_Genauigkeit:
Detektorfenster_Germanium:	Quellenmaterial_Nummer:
Abstand_Quelle:	Quellenmaterial_Dicke:

[8] This item is read since version 1.04.0000.

[9] This items are changed by the CalibrateEu152 or CalibrateEu152Ex function (see section 1.5). For more about the IDENTIFY setup file read the IDENTIFY user's manual.

[10] This item is read since version 1.03.0003.

```
FWHM_FWHM1:                          Absorbermaterial_Nummer:
FWHM_EXP1:                           Absorbermaterial_Dicke:
FWHM_Energie1:
```

3 Target directory

This chapter is only relevant for Microsoft Windows 95/98/NT/2000/ME/XP/Vista/7/8/10 and Microsoft Windows CE!

Quick IDENTIFY is delivered together with the IDENTIFY program[11]. It is copied into the program directory during the installation.

The installation routine of Pocket[12] IDENTIFY copies all files (identify.dll, r5.dat, standard.lib and identify.set) into the „\Program Files\Identify" directory. Normally, dynamic link libraries are copied into the „\windows" or application directory, but because of the three other files which can be edited or overwritten by the user and so that Pocket IDENTIFY can be used by several programs, the files are copied into an extra directory. If the dynamic link library is not within the „\windows" or application directory, you have to bind it during runtime explicitly (late binding). You find an example of late binding below. Of course, if you write your own program, you are free to copy all files into a directory you like.

Late binding example:

```
#include "identify.h"

HINSTANCE hIdentify;
RUNIDENTIFY pRunIdentify;

InitInstance()
{ ...
  hIdentify=LoadLibrary( _T("\\Program Files\\Identify\\Identify.dll"));
  if(hIdentify)
  { pRunIdentify=(RUNIDENTIFY)GetProcAddress(hIdentify,_T("RunIdentify"));
    if(!pRunIdentify)
    { FreeLibrary(hIdentify);
      hIdentify=NULL;
    }
  }
  ...
}

ExitInstance()
{ ...
  if(hIdentify) FreeLibrary(hIdentify);
  ...
}

Function()
{ ...
  CString strResult=pRunIdentify(lpulData,nChannels,fLiveTime,fRealTime);
  ...
}
```

[11] Please ask for a new version of the IDENTIFY program if Quick IDENTIFY is missing.

[12] Quick IDENTIFY for the Pocket PC

4 UNIX and Quick IDENTIFY

4.0 Introduction

We supply the source code for UNIX operating systems. In this way it is possible to compile and link the Quick IDENTIFY software to own programs on different UNIX operating systems.

The both directories within the "Unix" directory from the Quick IDENTIFY software package have to be copied to your unix system. Follow the instructions of the readme.txt file.

4.1 Programming

UNIX applications have to call InitIdentify (see section 1.8) at program start and ExitIdentify (see section 1.9) at program exit. These both functions serve the same tasks like the DllMain function *(see MSDN Library)* when the identify.dll *(for Microsoft Windows)* is loaded or unloaded.

The use of all functions and structures is identical to the dynamic link library for Microsoft Windows. They are described in chapter 1.

Do **not** modify the source code of Quick IDENTIFY! The only file that you can modify is rstrings.cpp[13]. If the line length of the strings are not to your taste, you can modify it by moving, adding or removing of „\r\n"-strings.

[13] Russian programmers should check the spelling of the Russian strings.

5 Revision history

The revision history lists the modifications and bug fixes since version 1.02.0003.

Version 1.03.0000

- New: Parameter *ulTextFlags* in functions `RunIdentifyFlag` and `RunIdentifyFlagWithBackground`. (see section 1.4)
- Fixed: In the previous revisions, uppercase characters in the library file name extension crashed the program.

Version 1.03.0001

- Fixed: A small difference between the IDENTIFY PC program and the Quick IDENTIFY library has been removed.

Version 1.03.0002

- Fixed: In the previous revisions, the functions `RunIdentifyFlag` and `RunIdentifyFlagWithBackground` wrote text to *lpszResult* above the limit set by the parameter *unLength.*

Version 1.03.0003

- Adapted: The software has been adapted to the version 1.07.0016 of the IDENTIFY PC program.

Version 1.04.0000

- Adapted: The software has been adapted to the version 1.08.0012 of the IDENTIFY PC program[14].

Version 1.05.0000

- Adapted: The software has been adapted to the version 1.09.0000 of the IDENTIFY PC program.

Version 1.06.0000

- Adapted: The software has been adapted to the version 2.00.0000 of the IDENTIFY PC program.

Version 1.06.0001

- Fixed: In the previous revisions, the file „fileextu.cpp" contained a bug. The destructor of `CFileExt` closed the file although it was already closed with the `Close()` member function.

Version 1.06.0002

- Adapted: The software has been adapted to the version 2.00.0001 of the IDENTIFY PC program.

Version 1.06.0003

- Adapted: The software has been adapted to the version 2.00.0016 of the IDENTIFY PC program.

Version 1.06.0004

- Adapted: The software has been adapted to the version 2.00.0027 of the IDENTIFY PC program.

Version 1.06.0005

- Adapted: The software has been adapted to the version 2.00.0029 of the IDENTIFY PC program.

[14] The peaks not assigned are listed now in the result string as far as the corresponding instruction is read from identify.set or the argument `ulTextFlags` of the function `RunIdentifyFlag` or `unIdentifyFlagWithBackground` is set to `IDF_WRITE_PEAKS_FOUND`.

Version 1.06.0006

- Adapted: The software considers the r6.dat[15] file now.

[15] The IDENTIFY PC program uses the r6.dat file already for a long time. Now the r6.dat file is also used by Quick IDENTIFY. The software always tries first to read r6.dat. Only if r6.dat cannot be read, it tries to read r5.dat. Make sure that the values within the identify.set file correspond to the right of these both files.