GOCHECK

A USER INTERFACE FOR CODECHECK IN MSVC

BY: Rijn Buve, CMG Research Centre Mission Critical Systems

DATE: September 23rd, 1999

TOPIC: GoCheck installation

1 WHAT IS GOCHECK?

GoCheck provides integration of CodeCheck™ with Microsoft Developer Studio (MSVC 97/98) through a Windows user interface that can be called from the MSVC Tools menu.

GoCheck was originally developed by Rijn Buve at the CMG Research Centre Mission Critical Systems for internal purposes. It has been made freely available for all CodeCheck users. CodeCheck has been tested thoroughly within CMG. If you experience any problems you may wish to contact Rijn Buve at the CMG Research Centre Mission Critical Systems (mcs@cmg.nl).

CodeCheck™, by Abraxas Software Inc, is a tool for the automatic checking of sources against coding standards (rules). It can be used stand-alone or from within MSVC to check source files. Unfortunately, without GoCheck this requires deep inside knowledge about both the CodeCheck command-line argument structure and the internal MSVC project settings.

For more information about CMG BV or the CMG Research Centre Mission Critical Systems, contact:

```
CMG BV
Att. Research Centre Mission Critical Systems
Post Office Box 187
2501 CD, The Hague
Netherlands
Phone:    +31 70-302-9302
Fax:      +31 70-302-9300
Internet: mcs@cmg.nl
```

For more information on CodeCheck, contact:

```
Abraxas Software, Inc.
Post Office Box 19586
Portland, OR 97219, USA
Phone:    +1 503-244-5253
Fax:      +1 503-244-8375
Internet: support@abxsoft.com
```

2 REQUIREMENTS

In order to be able to run GoCheck, you will need to have:

- **GoCheck.exe** - GoCheck installed in a directory specified in the PATH environment variable (usually C:\CodeCheck).
- **GoChkFlt.exe** - GoCheck Filter, installed in the GoCheck directory.
- **GoChkGo.exe** - GoCheck Go!, installed in the GoCheck directory.
- **GoCheck.cch** – The GoCheck rule file header, installed in the CodeCheck Rules directory (usually C:\CodeCheck\Rules), if you wish to use the additional GoCheck rule file functionality.

- **ChkNt.exe** - CodeCheck 8.01B1 or later.
- Microsoft Developer Studio 97 or 98.

GoCheck uses a fair amount of DOS environment space while executing CodeCheck (approximately 2000 bytes).

If (and only if) you experience problems with GoCheck, make sure you have enough environment space for **command.com** (Windows 95) by specifying the following line in **config.sys**:

```
device=c:\command.com /e:3000 /p
```

This is only necessary if you encouter “Out of environment space” warnings. You may wish to consult your systems administrator before doing so.

3 INTEGRATE GOCHECK IN MSVC

To integrate GoCheck into the MSVC Tools menu, do the following:

- Choose **Tools**/**Customize**/**Tools** and add a new item to the tools list.

  Item name: **&CodeCheck**

  Field **Command**: *<CodeCheck path>***\GoChkGo.exe**

  Field **Arguments**: *<CodeCheck path>***\GoCheck.exe**
  \"$(WkspDir)\$(WkspName).dsp\"
  \"$(FileDir)\"
  \"$(FileName)\"
  \"$(FileExt)\"

  Field **Initial directory**: *<empty>*

  Option **Use Output Window**: **Yes**

  Option **Prompt for arguments**: **No**

- CodeCheck can now be used from within MSVC by executing the CodeCheck menu item from the Tools menu, while the cursor is in a C++ source window.

4 CHANGES IN MSVC INCLUDE FILES

The include files as shipped with Microsoft Developer Studio 97 and 98 contain a number of known bugs. The MSVC compiler does not complain about these bugs. CodeCheck does, however.

You must therefore change the following lines in the include files (please notice that the precise location of the changes may vary with different releases of Developer Studio 97/98):

Microsoft Developer Studio 97

atlcom.h **line 1568** Add (forward declaration missing):

```
template <class T> class IConnectionPointContainerImpl;
```

atlcom.h **line 1826** Add (forward declaration missing):

```
template <class T> class CComEnum;
```

comdef.h **line 254** Missing parenthesis, add "**)**" at end of line:

```
#if defined (_COM_SMARTPTR_LEVEL2)
```

Microsoft Developer Studio 98

atlbase.h **line 5135** Missing semi-colon, add "**;**" at end of line:

```
T* p = &m_pBase[nElement];
```

atlcom.h **line 3985-3995** Incorrect order of declaration:
move to line 3650 just before:

```
template <…> class ATL_NO_VTABLE IdispEventSimpleImp…
```

atlcom.h **line 4388-4400** Incorrect order of declaration:
move to line 3995 just before:

```
template <…> STDMETHODIMP CcomEnumImpl<…>…
```

atlcom.h **line 4505-4517** Incorrect order of declaration:
move to line 4479 just before:

```
template <…> STDMETHODIMP IEnumOnSTLImpl<…>…
```

comdef.h **line 264** Missing parenthesis, add ")" at end of line:

```
#if defined (_COM_SMARTPTR_LEVEL2)
```

5 GOCHECK INTERNALS

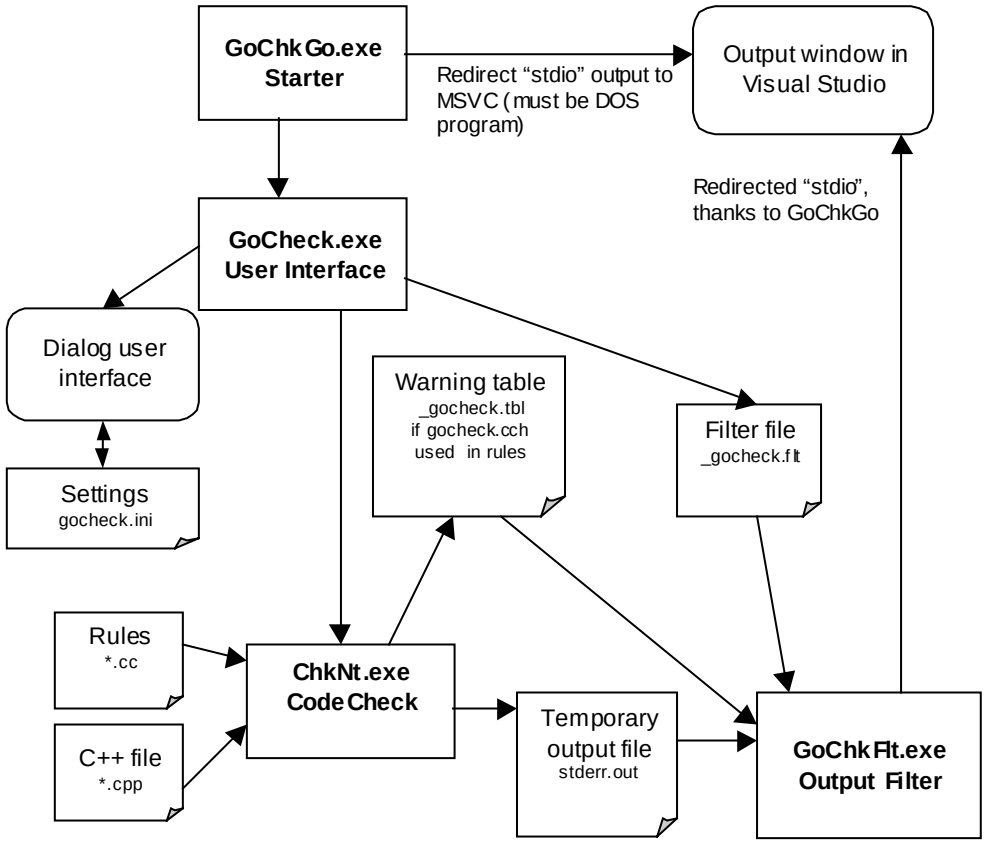
GoCheck consists of several small utilities working together. First of all GoChkGo is the "starter program". It starts the GoCheck user interface, while redirecting "stdout" to the MSVC output windows.

Second, GoCheck handles the user interface and reads its settings from the gocheck.ini settings file (found in the Windows directory). This settings file can be edited manually, if the user wishes to do so. Normally, this will not be necessary.

After pressing "OK" GoCheck generates a batch file to start CodeCheck and a filter file to be able to filter warning message according to the Filter settings page in GoCheck.

If "Run on OK" was selected, the automatically generated batch file is executed. This calls ChkNt.exe (the actual CodeCheck executable). The CodeCheck output is stored in "stderr.out". The stderr.out file, together with the message table and the filter file, are then processed by GoChkFlt, the output filter program. The output from GoChkFlt is redirected to the MSVC output window.

A schematic representation of this process is shown below.



6 GOCHECK.CCH RULE FILE HEADER

The **GoCheck.cch** rule file header contains additional functionality that can be used with GoCheck. To use the functionality, simple use "**#include <gocheck.cch>**" in your rule file.

- First of all, it predefines the following variables that can be used by any rule file:

```
int     i;
int     i1;
int     i2;
int     iLen;     // These variables can be freely used
int     ch;       // in rule files and do not have to be
int     ch1;      // defined by the user. They are NOT
int     ch2;      // reserved by this header file.
char*   str;      // Note that str, str1 and str2 are
char*   str1;     // pointers only. They do not automatically
char*   str2;     // allocate memory.
```

- The macro **WARN_DECL** is declared in gocheck.cch. Using this macro makes issuing warnings much easier in CodeCheck rule files. You only define the warning body text once and issue only the warning code (and, if necessary, any additional arguments) in your rule file. Besides, it offers you the ability to set limits to the number of times a warnings is shown in the CodeCheck output. All warnings will be defined once, at the top of your rule file, producing a clear table of warnings and warning limits. You can even (temporarily) disable warnings from the output without change insides of the rule file.
- The macros **IF_SYS_HEADER**, **IF_PRJ_HEADER**, **IF_NOT_SYS_HEADER** and **IF_NOT_PRJ_HEADER** are defined. These can be used to see if a source line originates from a system or project file. This also works if **#include <…>** is being used for project files, if the CodeCheck option **-V** is used, to specify where systems file are located. Example:

```
if ( lin_header )
{
  IF_NOT_SYS_HEADER
  {
    // Check project header file line.
  }
}
```

- The macro **HEADER** is defined to output a default header text for the rule file.
- The macros **INFO**, **INFO1**, …, **INFO3** are defined to output informational, non-filtered messages to the output.
- The variable **iTabSize** is set to the tab size field in the GoCheck Setting page (passed as CodeCheck option **-W**).
- The following defines are declared, missing from **check.cch**:

```
#define ACCESS_PUBLIC           0       // For dcl_access.
#define ACCESS_PROTECTED        1
#define ACCESS_PRIVATE          2

#define MEMBER_UNION            1       // For dcl_member.
#define MEMBER_STRUCT           2
#define MEMBER_CLASS            3
```

7 USING GOCHECK.CCH

To use **GoCheck.cch**, first include the appropriate rule file header:

```
#include <GoCheck.cch>
. . .
```

Below this line, declare all warnings once using the following syntax:

```
WARN_DECL( <num>, <limit>, <text> )
```

If the limit is -1, the warning is always issued. If it is 0 the warnings is never issued. If it is greater than 0, the warnings will be issued the specified amount of times.

```
WARN_DECL( 1010, -1, "Include file extensions should be '.h' or '.inl'" )
WARN_DECL( 1020, 9, "Source file should have .cpp extension" )
WARN_DECL( 1021, 1, "Do not check header files, check source files" )
WARN_DECL( 1050, 0, "Obligatory comment missing" )
. . .
```

The warning declarations list must be concluded by:

```
WARN_DECL_END()
. . .
```

At the start of your rule file, you may wish to output a header to the output windows, telling the user which version of the rule file is executed. You can use the HEADER macro for this:

```
if ( prj_begin )
{
    HEADER( "RBUVE_Stds.cc (Coding Standards) - 2.2 - 1999.05.03" );
}
. . .
```

In the actual rules, issue warnings using the "warn" command with the warning number only to issue warnings. The warning text will automatically be retrieved from the warning declaration section. If additional arguments are needed in the warning text, they will be added after the warning text. Specify them after the warning number in the warn command:

```
warn( 1050, "" )       Produces: Warning W1050: Obligatory comment missing
warn( 1050, "// File" ) Produces: Warning W1050: Obligatory comment missing: // File
```

Example:

```
//
// RULE 1:
//      Include files in C++ always have the file name extension ".h".
//      The #include "..." mechanism may not be used. The #include <...>
//      mechanism must be used instead.

#define PP_INCLUDE_ANGLE_BRACKETS   4

if ( header_name() )
{
    if ( pp_include == PP_INCLUDE_ANGLE_BRACKETS )
    {
        str = strrchr( header_name(), '.' );
        if ( !str )
        {
            warn( 1010, "%s", header_name() );
        }
        else if ( ( strcmp( str, ".h" ) != 0 ) &&
            ( strcmp( str, ".inl" ) != 0 ) )
        {
            warn( 1010, "%s", header_name() );
        }
        else
        {
            ;   // Empty.
        }
    }
}
. . .
```