CODE Conversion Tool

Reference Manual

Version 1.0

1. TABLE OF CONTENTS

2. INTRODUCTION

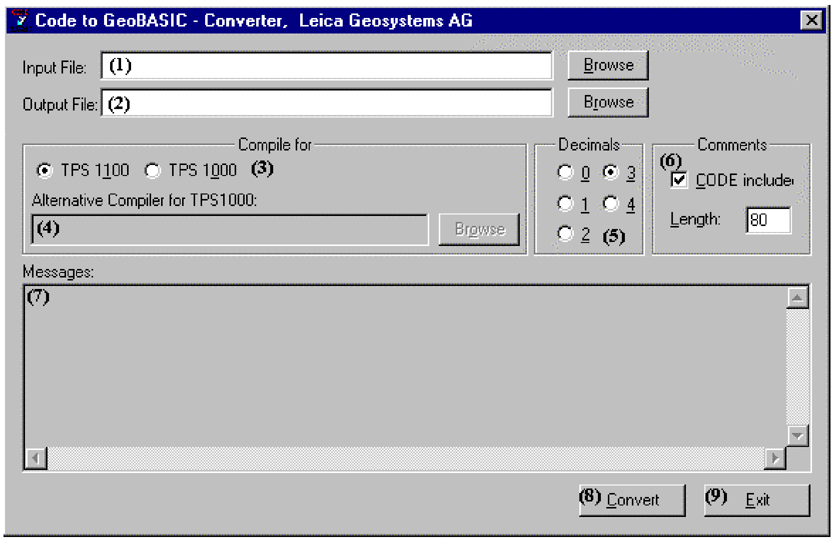
Many coding applications used on TPS1000 series instruments are programmed in the CODE development system which is by far not that flexible and powerful as GeoBASIC. To take advantage of GeoBASIC's functionality, CODE applications have to be converted to GeoBASIC applications. Of course, doing this manually takes a lot of time and the programmer has to take care of many things, like building an equivalent menu structure, simulating CODE's look and feel, and so on.

The Code_GB converter program is an easy to use tool that automatically converts CODE source code to equivalent GeoBASIC source code for TPS1100 and TPS1000 series instruments. The main advantages of this tool are:

- Data, saved in CODE with the `Rec` command, are saved in the exactly same format. Hence, programs working on saved data, do not need to be changed.
- The look and feel of converted applications stay the same.
- The converted GeoBASIC application can be extended manually.
- The converted GeoBASIC application can use GSI-8 and GSI-16 settings.

Note: Although the CODE to GeoBASIC converter transforms all CODE source into GeoBASIC source, it still lacks 100% compatibility. In certain circumstances it might be neccessary to "hand tune" the resulting GeoBASIC source.
Please notice that the converter's main intention is to convert to GeoBASIC source. Afterwards it will be expected that the programmer maintains the GeoBASIC source only.

3. Using the converter



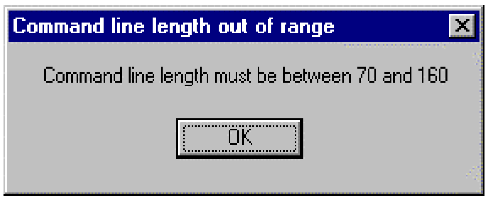
The figure shows the initial input mask of the converter. The name and path of the CODE file to convert can be entered into the field **(1)** manually or chosen using a file browser. The browser can be launched by pressing the "Browse" button on the right side of field **(1)**. In both cases automatically a proposal of the output's file name appears in the output file field **(2)**. The suggested name can also be changed either manually by entering the name into the field **(2)** or by pressing the Browser button to the right of field **(2)**.

In the section "Compile for" **(3)** one can choose the desired theodolite family. A change of this option affects both the conversion of the CODE program and the compiler used to compile the converted GeoBASIC file (See also 4.3.2). The GeoBASIC compiler for the TPS1100 family is embedded in the converter program whereas the compilation for TPS1000 invokes an external compiler. Choosing TPS1000 activates the input field and the "Browse" button for path and name of an alternative compiler **(4)**. Again the path and name of the compiler can be either entered manually or chosen using a file browser. If no alternative compiler is specified, Code_GB tries to invoke by default the compiler gbc_227.exe in the actual directory. If Code_GB fails to invoke the external compiler a message like the following is printed in the message window **(7)**:

```
GeoBASIC compiler messages:
ERROR: Cannot invoke compiler C:\xxx\gbc_227.exe
```

In the section "Decimals" **(5)** the desired number of decimals can be adjusted. The default value is 3 decimals. Unlike CODE GeoBASIC does not use the theodolites' settings for the number of decimal digits. Hence a change of this setting requires a new conversion of the CODE program.

The option "CODE includes" in the section "Comments" **(6)** enables or disables embedded, outcommented CODE lines in the converted GeoBASIC code. "Length" specifies the length of such comment lines, hence this field will only be activated if "Code includes" is checked. The length of comment lines must be between 70 and 160, otherwise this message box will appear:


Command line length out of range

Command line length must be between 70 and 160

To start the conversion process, press the "Convert" button **(8)** . During the conversion and compilation process, the "Convert" button is deactivated. All messages concernig conversion and compilation are displayed in the message window **(7)** (see also 4.2). To end the program press the "Exit" button **(9)**.

Note:	This tool does not need a hardware protection key, as it is necessary for the GeoBASIC developement environment. Be aware of the fact, that, if one has to "hand tune" the converted GeoBASIC source, then it is necessary to buy the GeoBASIC developement environment to compile this changes into the application.

4. THE CONVERSION PROCESS

This section only covers the conversion (CODE to GeoBASIC source) but not the compilation process (GeoBASIC source to GeoBASIC application)! For further information about GeoBASIC errors please refer to the GeoBASIC manual.

The conversion tool reads CODE statements from an input file and writes the converted input to an output file. A message will be generated on the occasion of a syntax error or a conversion problem. (See also chapter 4.2). In the former case the conversion process will be stopped and no output file will be generated. In the latter case the conversion process continues (With one exception: A menu entry that references to an unknown procedure/menu) . This semantical error also causes the converter to stop. See chapter 4.2.2 for more information).

After the successful conversion of a CODE source file, a short statistic of the conversion process is shown in the message window.

To use the converted output file on theodolites, the file has to be compiled successful with the GeoBASIC compiler. That will be done after the conversion automatically.

An output example of a successful conversion:

```
Conversion process started
Line 4 : WARNING: String constant too long, -> Text cut off
CODE converted successfully.

Statistics:
   1 warning(s)
   No informationals
```

Output example of an unsuccessful conversion :

```
Conversion process started
Line 4 : WARNING: String constant too long, -> Text cut off
Line 5 : Informational: Variable "if" had to be renamed
LINE 21 : ERROR: Syntax error detected!
Abnormal program termination
```

Every warning or informational, found by the converter, is also added to the output file as comment. Such comments are marked visually with ‘#’ characters. For example, the above warning looks in the output file like following :

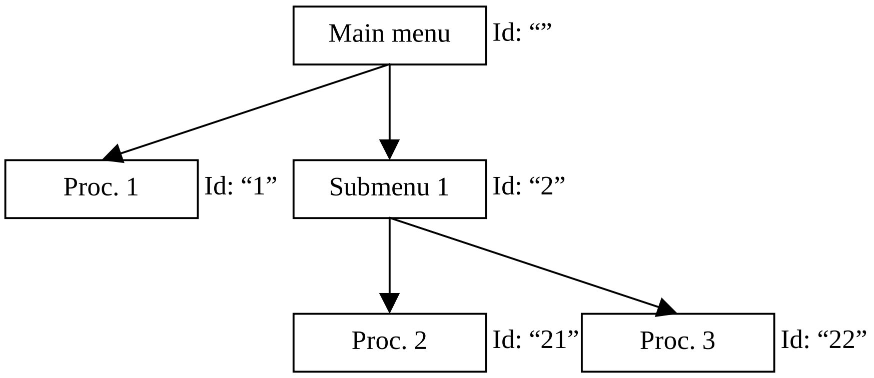
```
'##############################WARNING: String constant too long, -> Text
cut off
CONST DEFAULT As STRING30 = "000000000000000"
```

4.1 STRUCTURE OF A CONVERTED PROGRAM

The main task of the converted program is to simulate the CODE program’s menu handling. In CODE a menu consists of one to nine menu entries. A menu entry can be either a procedure or another menu. The menu structure, defined by the combination of procedures and submenus, is called a menu tree. As a shortcut we call a menu or a procedure a node.

Each of these nodes can be identified by a unique Id, which is needed to keep track of the position in the menu tree during runtime. The Id of a node consists of the digits that have to be pressed in order to reach this menu.

Consider following simple menu tree:



For example, to execute Proc. 2 one has to choose menu entry 2 in the main menu and then menu entry 1 in the Submenu 1.

In general, Code_GB generates two kinds of functions, one to handle a CODE-menu (see section 4.1.1) and the other to handle a CODE-procedure (see section 4.1.2). For each CODE-menu and CODE-procedure a corresponding function will be generated in GeoBASIC (see section 4.1.3) .

To make the whole application run, some help functions have to be generated too.

The general structure of a converted program can be described as following:

1. Header with some information
2. Translation of Codename, constants and variables (in order of appearance).
3. Declaration of help constants and variables with the prefix cgb_ (e.g. cgb_dspWidth).
4. Declaration of help procedures.
5. Conversion of menu entries representing CODE procedures.
6. Conversions of menu entries representing menus. The menus are ordered in descending level of nesting depth.

4.1.1 Description of a converted menue

The name of a menu function depends on the name used in the CODE program. The parameter, passed to the function, contains the call path to the menu (used for `C-Path`[1]) and the Id of the procedure to start with (only if `C-Next` was used).

The menu function basically consists of one main loop to process the menu. This loop terminates only if the menu has to be left.

At first, inside this loop, it will be determined if a specific menu entry preselection has to be made and if a submenu has to be entered. A preselection will be made visible by highlighting the corresponding menu entry. Initially the first menu entry will be preselected by default. If another submenu has to be entered, control will be given to this function, instead of displaying the menu.

After a valid selection of a menu entry, the corresponding function of the chosen menu entry is called. This function returns, if a `C-Next` command has to be simulated, with the desired Id of the node in the menu tree. Otherwise “” is returned only.

The last decision to be made in the menu function is if this function should be left or not.

Reasons to leave are

- TPS1000: Button ‘ESC’ was pressed. (Saved in the global variable cgb_exit, see sections 4.1.3, 4.3.2).
- TPS1100: Button ‘Shift-F6’ was pressed. (Saved in the global variable cgb_prgFlow, see sections 4.1.3, 4.3.2)
- The next node to go to (with `C-Next`) cannot be reached from the actual node.
- There is no next node to be executed.

4.1.2 Description of a converted procedure

A converted procedure follows a simple scheme. At the beginning a text dialog is created . If the program was compiled for TPS1100 the “F1”-key is added. This key is a substitute for the “CONT” key of TPS1000 series theodolites. Hereafter each original CODE command will be converted into one or more GeoBASIC statements. At the bottom of the function the text dialog gets deleted and the function’s return value gets assigned. The return value complies to the node Id that has to be activated next. If the program has to be left, then the return value will be “”.

[1] C- is used as abbreviation for CODE command. E. g. “`C-Path`“ substitutes “CODE command `Path`”.

4.1.3 Description of help procedures and variables

Please notice that, for readability, all automatically generated identifiers get a prefix. Constants and variables get the prefix “cgb_” and functions “Cgb_”

Global constants:

Name of constant	Description
cgb_dFix	Contains the number of decimals used. The value of this constant depends on the conversion settings (see also section 3)
cgb_dspWidth	Maximum number of characters per line supported by CODE.
cgb_dX	Starting column of display in CODE.
cgb_dY	Starting line of display in CODE.
cgb_isDouble	Indicator for parameter type used in Cgb_put.
cgb_isString	Indicator for parameter type used in Cgb_put.
cgb_noPutformat	Special value for format used by Cgb_put that indicates that no reformat of parameter is needed.
cgb_gsi_rec8	Standard width of fields in GSI-8 mode (see also 4.3.1).
cgb_gsi_rec16	Standard width of fields in GSI-16 mode (see also 4.3.1).
cgb_exec	Used for cgb_prgFlow: Continue program. TPS1100 only!
cgb_exitProc	Used for cgb_prgFlow: Leave actual procedure. TPS1100 only!
cgb_endProg	Used for cgb_prgFlow: Leave the program. TPS1100 only!

Global variables:

Name of variable	Description
cgb_wiEntry	Data structure needed to get and set WI values.
cgb_recMask	Array containing WI Ids to record (Set new for each CODE Rec command, declared global as it has to be initialized)
cgb_startNext	Variable containing the Id of the starting node used on next start of program.
cgb_nextPath	Variable containing the Id of the next node to be executed.
cgb_exit	Flag that indicates stop of program (‘ESC’ was pressed during input). TPS1000 only!
cgb_inpNumLen	Length of a numerical input field, depends on GSI setting.
cgb_inpStrLen	Length of a textual input field, depends on GSI setting.
cgb_wiLength	Length of a WI data field, depends on GSI setting.
cgb_inpNumMin	Minimal number allowed at input, depends on GSI setting.
cgb_inpNumMax	Maximal number allowed at input, depends on GSI setting.
cgb_prgFlow	Indicator of the actual program flow state. This variable can only contain cgb_exec, cgb_exitProc or cgb_endProg. TPS1100 only!

Help procedures:

The help procedures simulate the behaviour of the original CODE application. The first action of every help procedure is to check for the global variable cgb_exit on TPS1000 or the variable cgb_prgFlow on TPS1100 and react properly on it.

Name of help procedure	Description
Cgb_ZeroTrim	This procedure is a routine needed often in help procedures. It removes all leading zeroes of a string.
Cgb_StrCopy	Simulation of the `C-Strcpy`. This routine still uses the old way of retrieving single characters (not by indexing the string) to make it compatible for older GeoBASIC versions.
Cgb_StrCopy_16	A special version of Cgb_StrCopy to help prevent strcopy problems in GSI-16 mode (see also section 4.3.1).
Cgb_PrintStr	Prints a given string. The parameter allignPos is responsible for different handling of string constants and variables that exceed the right side of the display.
Cgb_PrintVal	Reformats a value to the CODE style and prints it, using the Cgb_Print procedure.
Cgb_PrintWi	Handles the output for any value saved in a WI-record depending on its data type of the WI (Double, signed ASCII, ASCII). Also uses Cgb_PrintStr for output.
Cgb_ClrDsp	Deletes rectangular sections of the display by printing strings filled with spaces. This procedure shows another behavior than CODE if negative coordinates are passed (should never happen anyway).
Cgb_Put	This procedure is responsible for writing values to a WI. It distinguishes between double- and stringvalues. To be able to simulate the various formatting possibilities of double values exactly, they have to be converted to a string and saved as GSI_SIGNED_ASCII.
Cgb_InputStr	This procedure is responsible for input of strings. To avoid incompatibilities (e.g. in Cgb_strcopy) all strings are filled up with leading literal zeroes just like in CODE. It is also checked, if ‘ESC’ was pressed on prompt which results in ending the program by setting the global variable cgb_exit to true.
Cgb_InputVal	This procedure manages the input of numeric values. Equal to Cgb_InputStr it is checked whether ‘ESC’ was pressed or not.
Cgb_Pause	Waits for the keys ‘F1’ (CONT), ‘ESC’ or ‘SHIFT-ESC’. If latter two are pressed, cgb_exit is set to true (program termination).
Cgb_Rec	Saves up to 8 WIs (limitation of CODE). If less than eight WI ids are passed in the original `C-Rec`, the converter fills up the remaining actual parameters with `GSI_ID_NONE` automatically.
Cgb_DspPath	This procedure receives the full path and prints the biggest possible path on screen. The actual path is known for all visited nodes in the menu tree.
Cgb_Beep	Beeps short if parameter = 0, long otherwise.
Cgb_Delay	Simulates the `C-Pause`.

4.2 MESSAGES OF THE CONVERSION TOOL

The converter assumes the CODE application, that has to be converted, to be syntactically and semantically correct. Nevertheless, the converter program recognizes many semantic errors and, moreover, can cope with some of them. The following chapters explain the three different kinds of errors and how the converter reacts on them.

4.2.1 Syntactical Errors

The worst error, that can happen, is a syntactical error. This kind of error always causes the converter to stop working immediately. If the converter encounters a syntactical error a message will be written on the output window with the number of the erroneous line in the CODE source. An example for such a message is:

```
Line 42 :ERROR: Syntax error detected !
Abnormal program termination
```

The reason for syntactical errors may be unknown keywords, forgotten parameters/semicolons, invalid number formats, type mismatches and so on. Therefore, we recommend to convert CODE programs only, that can be compiled successfully with the CODE compiler. For more precise information about the syntax of CODE, please refer to the CODE manual.

4.2.2 Sematical Errors

Semantical errors describe those errors where constructs do not follow the sematical rules. For example a string assignment to a real variable may be syntactical correct but is semantical incorrect. These warnings should not be ignored because they can change the behavior of the converted program or even lead to a GeoBASIC compiler error. The best way to avoid warnings is to convert only programs that are compileable without warnings in CODE.

With one exception, no semantical error leads to the abortion of the converter. This happens if a menu tries to invoke an unknown procedure/submenu. In this case, the converter can not generate a correct menu tree and aborts with an error message that is printed to the output window. Such an error message looks like the following:

```
ERROR: Submenu / Procedure  StatCoorXX was never declared but used in menu
declaration
```

The converter specifies the erroneous submenu/procedure name but can not give line information of this error. Nevertheless finding the faulty line in the source code is not hard as almost all text editors support finding a specific text.

The way the compiler handles all other warnings are very different:

Some semantic errors will be ignored in such a way that the converter uses a default rule for conversion:

- Convert the statement, if the semantic error is not serious and a default rule is implemented in the converter. E.g. the command `DSP/ac/0/$blah;` is not compileable with CODE.exe if `ac` is declared as a real variable. Nevertheless, an expression like this can be converted and will be compiled without an error by the GeoBASIC compiler.
- Do not convert it and generate a warning, if, for example, the statement `DSP/ac/0/$blah;` uses the undeclared variable `ac`.

Some semantic errors lead to the following behavior:

- Convert the command and generate a warning. This is especially valid for `C-Next` statements. If, for instance, a statement like `Next/MAINMENU /ab;` is found and `ab` is undeclared or declared in an invalid way, then this statement will be replaced by a correct CODE source statement internally. For example, like `Next/MAINMENU/1;` Afterwards the Code source statement will be converted and a warning is written just above the converted command into the output file.

Also a warning will be generated, if a text string is longer than 15 Characters or the textual representation of a real value is longer than 12 digits (including decimal point). In this case the result has to be truncated, following the rules of CODE.

Furthermore, some semantic errors exist, which can not be recognized and/or handled by the converter program. If, for instance, a string variable is assigned to a real variable, the assignment will be converted. Of course, this illegal statement will be detected by the GeoBASIC compiler, which generates an error message for it.

A small set of commands, which produce warnings in CODE, cannot be recognized correctly by either the converter program or the GeoBASIC compiler. Such commands may result in a different behavior of the converted GeoBASIC programs. For example the ‘invalid’ clearance of the display `clrdsp/-2/-5 /20/16;` will not be handled by the converter. This can be avoided only, if the user inputs valid CODE statements into the converter program.

To find generated warnings easier, which will be inserted into the converted GeoBASIC source code, they will be marked visually with ‘#’ characters. For example:

```
'-------------(line:65)    Dsp/v_real01/0/consta;
'###Can’t convert Statement (consta not declared?)
```

4.2.3 Informationals

Informationals describe necessary changes, that had to be made by the converter in order to make the generated GeoBASIC file compileable.

The difference, between a warning and an informational, is that a converted statement with a warning may alter the behavior of the resulting program. Whereas conversions of CODE statements, that produce informationals do never alter the behavior of the resulting program. Informationals only tell the user which changes had to be made by the converter.

Informationals can be caused by:

- An identifier (e.g. variable name, procedure name,...) that resembles a keyword in GeoBASIC. Like most programming languages, GeoBASIC does not allow identifiers that match any keyword of its language. Therefore the converter renames such identifiers by adding two underscores to the end of the identifier. For instance an identifier called `Sub` will be renamed to `Sub__` on every occurrence.
- An identifier starting with underscore(s). An Underscore must not be a first character of an identifier in GeoBASIC but is allowed in CODE. Hence underscores are moved to the end of the identifier (e.g. `_i_count` gets `i_count_`)

The informational will not be printed on every occurrence of a renamed identifier but only at the identifier’s first occurrence.

Similar to warnings, informationals are marked visually with a ‘#’.For instance:

```
'---------------------------(line:30)sub: String;
'####Informational: Variable sub had to be renamed
DIM sub__ AS STRING30
```

4.3 FEATURES OF THE CONVERSION TOOL

4.3.1 GSI 8 and GSI 16 support

The theodolites’ built-in database functionality supports two different recording formats: On one hand the GSI8 format where 8 characters are used to store texts and numbers (in file) and on the other hand GSI16 a 16 byte recording format.

A CODE file, converted by Code_GB, however, manages to adapt to GSI settings automatically at run time. This means changing the recording format does not require a change or a new translation of the original CODE file.

Anyway, due to display and other limitations, a CODE program running with GSI8 may have a slightly different behavior than a converted GeoBASIC file, running in GSI16 mode.

Differences between CODE-GSI8 and resulting GeoBASIC-GSI16 are:

- A converted `C-Inp` string input prompt uses a size of 16 characters (instead of 8) and an input prompt for real variables uses a size of 18 (instead of 10) characters.
- A converted `C-Dsp` of string type WI values will be cut off on the right side, if label and output text exceed display line length (currently 29). Only the display output of the value will be shortened not the string itself.
- A converted `C-Dsp` output of a real or string value will be aligned differently. As a side effect, if the string value results from an `inp` command, those values are displayed right aligned to 10 (8 for strings) characters. To keep

up the alignment for bigger numbers in GSI16, these values are aligned to 18 (16 for strings) characters. Of course, this will result in a different outlook of the display.

- A converted `C-Dsp` of real type WI values does not contain the usual padding ‘0’ characters, because this would lead in a cut off of the displayed string. Instead the non-padded version will be used. Again this does not influence the actually stored string, but just the display output.
- Strings that received their values via `C-Inp` (NOT by assignment) are saved in accordance to CODE specification right alligned. This may lead to problems with `C-strcopy` due to changed indices in GSI-16 mode. Consider following example:

```
str1 := “pipe :“;
inp/0/0/str2; (* input eg. 12 *)
strcopy/str1/6/str2/6/2;
```

GSI-8 result „pipe :12“
GSI-16 result „pipe :0000000000“ because str2 contains „0000000000000012“.

A solution to this problem is the new CODE command `strcopy_16`. The special purpose of StrCopy_16 is that it intreprets the start indexes in such a manor that the rightmost 8 characters of the source and target string get the character indexes 0 to 7.
For example Situation 1: GSI8
target = "01000000"
source = "00000002"

```
StrCopy_16/target/ 7/source/ 7/1;
```

results in "01000002"

Situation 2: GSI16
target = "0100000000000000"
source = "0000000000000002"

```
StrCopy_16/target/7/source/ 7/1;
```

results in "01000000<u>00000002</u>". (Only the underlined characters are affected).

Hence the same statement results, depending on the recording format, in two different strings. Note that assigned strings (eg. `str := “abc“)`are NOT affected by any index shift!

4.3.2 Differences between TPS1000 and TPS1100

A design goal of the code converter was not to break up with standard look and feel of the TPS1000 and TPS1100. Therefore conversion of CODE programs for different series lead to minor differences in the usage of converted programs:

Key	TPS1000	TPS1100
ESC	During `c-pause` or `c-inp`: Exit program Pressed in menu: Jump back one hierarchy level in the menu structure.	During `c-pause` or `c-inp`: Exit procedure and jump back to menu. Pressed in menu: Jump back one hierarchy level in the menu structure.
SHIFT-ESC	Same as “ESC”	No function
SHIFT-F6	No function	Exit program (no matter when pressed)
CONT	Continue	Not available
F1	Not available	Continue (Substitute for CONT-Key on TPS1000)