Converter from GEL to PRACTICE

Converter from GEL to PRACTICE

Version 18-Nov-2020

Introduction

This document describes using General Extension Language to PRACTICE Converter. The executable file can be found in the TRACE32 installation directory under ~~/demo/tools/gel_converter.

The General Extension Language (GEL) is an interpretive language similar to C that lets you create functions to extend Code Composer Studio's usefulness. This converter allows you to convert this language into PRACTICE cmm scripts, which can be used directly in TRACE32.

Brief Overview of Documents for New Users

Architecture-independent information:

- **"Debugger Basics - Training"** (training_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **"T32Start"** (app_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **"General Commands"** (general_ref_*<x>*.pdf): Alphabetic list of debug commands.

Architecture-specific information:

- **"Processor Architecture Manuals"**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:

  - Choose **Help** menu > **Processor Architecture Manual**.

- **"OS Awareness Manuals"** (rtos_*<os>*.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

Launching Converter

Format:	**converter [-aARCH_NAME] [-d]** *<input_file> <output_file>*
[-aARCH_NAME]	Optional parameter. Selects target architecture. This option enables CPU register names recognition in GEL code for selected target. Valid architectures are: C620X, C6201, C621X, C64X, C6416, C64X+, C670X, C671X, C6713, C54X, C55X, C5510. If no architecture is selected, register names are treated as target symbols, except of register names that are used as local variables in functions.
[-d]	Optional parameter. Disables errors if not supported GEL functions are used and marks places in CMM output file with commented functions names.
[-w]	Optional parameter. Redirects output from GEL_TextOut and GEL_TargetTextOut functions to default TRACE32 AREA window instead of named AREA window (‘windowName’ parameter)
<input_file>	Input GEL file.
<output_file>	Output CMM file

Using @prog, @data and @io

Only following formats are supported by converter when using @prog, @data and @io GEL constructions.

```
*(sign_modifier data_type *)constant@suffix

*(sign_modifier           *)constant@suffix
```

Where:

```
sign_modifier    = unsigned | signed

data_type        = char | void | short | int | long | long long

constant         = hexadecimal_number | decimal_number

suffix           = prog | data | io
```

Additional simplified format can be used:

```
*constant@suffix
```

NOTE: Integer is always treated as 32-bit data type.

Using Menuitem, Hotmenu, Dialog and Slider

`slider` keyword is not supported. Functions marked by `slider` are discarded while converting GEL script to PRACTICE cmm script.

Only following construction is supported:

```
menuitem "menu_item_name";

hotmenu HotmenuFunctionName()
{
     …
}

dialog DialogFunctionName(dialog_function_parameters…)
{
     …
}
```

There can be more than one `dialog` or `hotmenu` after `menuitem`. Order of `dialog` and `hotmenu` is unrestricted. There cannot be any other functions than `hotmenu` or `dialog` after `menuitem`. If there are any, it is assumed to be the end of `menuitem` block.

Recognizing Types of Identifiers in GEL

Following types of identifiers may appear in GEL language:

- Local variables
- Global variables
- Target symbols
- Target CPU register names

Following order is used when recognizing type of identifiers in GEL script.

1. If architecture is defined - CPU register names.
2. Local script function variables / Global script variables.
3. Target symbols.

Functions Parameters

Functions parameters (except target symbols as parameters) are passed to function as in GEL. Target symbols are passed to function as their numeric values, not the names.

Preprocessor

Converter supports `#define` GEL directive, however `#define` macro cannot be used before its declaration.

Callback GEL Functions

GEL callback functions are not supported, however You can call them directly like other script functions.

Built-in GEL Functions

Converter and PRACTICE supports some subset of Built-in GEL functions with some limitations. Not supported functions calling causes converting error. Errors can be disabled by option **-d**

Supported functions:

```
GEL_AsmStepInto()                   GEL_Reset()
GEL_AsmStepOver()                   GEL_Run()
GEL_BreakPtAdd()                    GEL_RunF()
GEL_BreakPtDel()                    GEL_SrcDirAdd()
GEL_BreakPtDisable()                GEL_SrcDirRemoveAll()
GEL_BreakPtReset()                  GEL_SrcStepInto()
GEL_CloseWindow()                   GEL_SrcStepOver()
GEL_Exit()                          GEL_StepInto()
GEL_Go()                            GEL_StepOut()
GEL_Halt()                          GEL_StepOver()
GEL_HWBreakPtAdd()                  GEL_SyncHalt()
GEL_HWBreakPtDel()                  GEL_SyncRun()
GEL_HWBreakPtDisable()              GEL_SyncStepInto()
GEL_HWBreakPtReset()                GEL_SyncStepOut()
GEL_MemoryFill()                    GEL_SyncStepOver()
GEL_MemoryLoad()                    GEL_System()
GEL_OpenDisassemblyWindow()         GEL_UnloadAllSymbols()
GEL_PatchAssembly()                 GEL_WatchDel()
GEL_RefreshWindows()
```

Partial supported functions:

GEL_Animate()	Runs the program until a breakpoint is encountered. At the breakpoint, execution stops. After updating windows, program resumes its execution until next breakpoint.
GEL_IsInRealTimeMode()	Always returns TRUE.
GEL_Load()	'CpuName' and 'boardName' parameters are not supported
GEL_LoadGEL()	Supports only PRACTICE scripts. GEL scripts need to be converted to PRACTICE scripts first.
GEL_MemorySave()	'io_Format' and 'append' parameters are not supported. Memory block is always saved in hexadecimal format and file is always overwritten.
GEL_OpenMemoryWindow()	Parameters: 'title', 'qValue', 'ref_buffer_on', 'ref_buffer_start', 'ref_buffer_end', 'ref_buffer_auto_update', 'track_expression', 'bypass_cache', 'highlight_cache_diffs' are not supported.
GEL_OpenWindow()	'WindowType' and 'maxLines' parameters are not supported.
GEL_SymbolAdd()	'CpuName' and 'boardName' parameters are not supported.
GEL_SymbolLoad()	'CpuName' and 'boardName' parameters are not supported.
GEL_SymbolRemove()	Only filename is supported - file path is discarded. 'CpuName' and 'boardName' parameters are not supported.
GEL_TargetTextOut()	Only 'startAddress', 'page' and 'windowName' parameters are supported.
GEL_TextOut()	'textColor', 'lineNumber' and 'appendToEnd' parameters are not supported.
GEL_WatchAdd()	'Label' parameter is not supported.

Using PRACTICE Commands from GEL Script

There is possibility to execute PRACTICE command directly from GEL script by using following syntax:

```
//!PRACTICE(practice_command);
```

Above construction must be placed in an empty line.

Converter-specific Reserved Words

Converter is using following subset of identifiers:

```
__V0, __V1, __V2, …
__L0, __L1, __L2, …
```

This identifiers cannot be used as labels, local/global variables, symbols and function names in GEL scripts.

Target CPU Register Names

TARGET	REGISTERS
C620X, C6201, C621X	A0-A15, AMR, B0-B15, CSR, EM, ER, IER, IFR, IRP, ISTP, NRP, PC
C64X, C6416	A0-A31, AMR, B0-B31, CSR, DIER, EM, ER, GFPGF, IER, IFR, IRP, ISTP, NRP, PC
C64X+	A0-A31, AMR, B0-B31, CSR, DIER, DNUM, ECR, EFR, EM, ER, GFPGF, GPLYA, GPLYB, IER, IERR, IFR, ILC, IRP, ISTP, ITSR, NRP, NTSR, PC, REP, RILC, SSR, TSCH, TSCL, TSR
C670X, C671X, C6713	A0-A15, AMR, B0-B15, CSR, FADCR, FAUCR, FMCR, IER, IFR, IRP, ISTP, NRP, PC
C54X	A, AR0-AR7, B, BK, BRC, DP, IFR, IMR, IPTR, PMST, REA, RSA, SP, ST0, ST1, T, TRN, XPC
C55X, C5510	AC0-AC3, BK03, BK47, BKC, BRC0, BRC1, BRS1, BSA01, BSA23, BSA45, BSA67, BSAC, CFCT, CSR, DBIER0, DBIER1, IER0, IER1, IFR0, IFR1, IVPD, IVPH, MDP, MDP05, MDP67, PC, PDP, REA0, REA1, RETA, RPTC, RSA0, RSA1, ST0-ST3, T0-T3, TRN0, TRN1, XAR0-XAR8, XCDP, XDP, XSP, XSSP

Troubleshooting

PROBLEM:	SOLUTION:
TRACE32 displays error: “unknown area”	GEL_TextOut() or GEL_TargetTextOut() function was used with ‘windowName’ parameter and TRACE32 AREA window with that name is not yet created. Either create AREA window with GEL_OpenWindow() or use converter option “-w” to redirect GEL_TextOut() and GEL_TargetTextOut() to single default TRACE32 AREA window.