

hsk-libs-scripts

270

Generated by Doxygen 1.8.12

Contents

- 1 HSK XC878 μ C Library Build Scripts 1**
 - 1.1 Compatibility 1
 - 1.2 Layout 1

- 2 File Index 3**
 - 2.1 File List 3

- 3 File Documentation 5**
 - 3.1 Makefile File Reference 5
 - 3.1.1 Detailed Description 5
 - 3.2 Makefile.include File Reference 6
 - 3.2.1 Detailed Description 6
 - 3.3 scripts/awk2doxygen.awk File Reference 6
 - 3.3.1 Detailed Description 7
 - 3.3.2 Function Documentation 7
 - 3.3.2.1 filter14() 7
 - 3.3.2.2 debug() 7
 - 3.3.2.3 filter38() 8
 - 3.3.2.4 initDoc() 8
 - 3.3.2.5 genFunction() 8
 - 3.3.2.6 filter84() 8
 - 3.3.2.7 filter94() 9
 - 3.3.2.8 filter103() 9
 - 3.3.2.9 filter115() 9

3.3.2.10	filter178()	9
3.3.2.11	filter185()	10
3.3.2.12	filter196()	10
3.3.2.13	filter203()	10
3.3.2.14	filter219()	10
3.3.2.15	filter226()	10
3.3.2.16	filter241()	11
3.3.2.17	filter254()	11
3.4	scripts/build.awk File Reference	11
3.4.1	Detailed Description	12
3.4.2	Function Documentation	13
3.4.2.1	filter118()	13
3.4.2.2	verbose()	14
3.4.2.3	verbose_values()	14
3.4.2.4	verbose_keys()	14
3.4.2.5	esc()	15
3.4.2.6	compact()	15
3.4.2.7	flags()	15
3.4.2.8	src_to_obj()	16
3.4.2.9	src_to_origin()	16
3.4.2.10	obj_to_bin()	16
3.4.2.11	linklist()	17
3.5	scripts/cstrip.awk File Reference	17
3.5.1	Detailed Description	17
3.5.2	Function Documentation	18
3.5.2.1	filter18()	18
3.5.2.2	filter44()	18
3.6	scripts/dbc.sh File Reference	18
3.6.1	Detailed Description	18
3.7	scripts/dbc2c.awk File Reference	19

3.7.1	Detailed Description	21
3.7.2	Environment	21
3.7.2.1	DEBUG	21
3.7.2.2	TEMPLATES	21
3.7.2.3	DATE	21
3.7.3	Value Tables	21
3.7.4	Templates	22
3.7.4.1	Special Attributes	22
3.7.4.2	Inserting Data	22
3.7.4.3	header.tpl	23
3.7.4.4	file.tpl	23
3.7.4.5	sigid.tpl	23
3.7.4.6	ecu.tpl	24
3.7.4.7	msg.tpl	24
3.7.4.8	siggrp.tpl	24
3.7.4.9	sig.tpl	25
3.7.4.10	timeout.tpl	26
3.7.4.11	enum.tpl	27
3.7.5	Function Documentation	27
3.7.5.1	filter329()	27
3.7.5.2	filter439()	27
3.7.5.3	error()	27
3.7.5.4	warn()	28
3.7.5.5	debug()	28
3.7.5.6	buffer()	28
3.7.5.7	fetchStr()	28
3.7.5.8	fetch()	29
3.7.5.9	whole()	29
3.7.5.10	strip()	29
3.7.5.11	getContext()	30

3.7.5.12	getUniqueEnum()	30
3.7.5.13	fsm_discard()	31
3.7.5.14	fsm_ecu()	31
3.7.5.15	fsm_enum()	31
3.7.5.16	fsm_sig_enum()	32
3.7.5.17	fsm_env()	32
3.7.5.18	fsm_env_data()	32
3.7.5.19	fsm_msg()	33
3.7.5.20	fsm_sig()	33
3.7.5.21	fsm_comment()	34
3.7.5.22	fsm_attrrange()	34
3.7.5.23	fsm_relattrrange()	34
3.7.5.24	fsm_attrdefault()	35
3.7.5.25	fetch_attrval()	35
3.7.5.26	fsm_attr()	35
3.7.5.27	fsm_relattr()	36
3.7.5.28	fsm_symbols()	36
3.7.5.29	fsm_tx()	36
3.7.5.30	fsm_siggrp()	37
3.7.5.31	fsm_start()	37
3.7.5.32	filter1434()	37
3.7.5.33	euclid()	37
3.7.5.34	rationalFmt()	38
3.7.5.35	rationalIN()	38
3.7.5.36	rationalID()	38
3.7.5.37	rational()	40
3.7.5.38	filter()	40
3.7.5.39	tpl_line()	41
3.7.5.40	template()	41
3.7.5.41	setTypes()	42

3.7.5.42	sigident()	42
3.7.5.43	siggrpident()	42
3.7.5.44	msgid()	43
3.7.5.45	msgidext()	43
3.7.5.46	filter1841()	43
3.8	scripts/depends.awk File Reference	44
3.8.1	Detailed Description	44
3.8.2	Modes of Operation	44
3.8.3	Environment	45
3.8.3.1	SUFFIX	45
3.8.4	Function Documentation	45
3.8.4.1	testf()	45
3.8.4.2	rescape()	45
3.8.4.3	sescapex()	46
3.8.4.4	extract()	46
3.8.4.5	any()	46
3.8.4.6	compact()	47
3.8.4.7	filter184()	47
3.9	scripts/file2doxygen.awk File Reference	48
3.9.1	Detailed Description	48
3.9.2	Function Documentation	48
3.9.2.1	filter13()	48
3.9.2.2	filter22()	48
3.9.2.3	filter31()	49
3.10	scripts/filter.sugar.awk File Reference	49
3.10.1	Detailed Description	49
3.10.2	Function Documentation	49
3.10.2.1	filter10()	49
3.10.2.2	filter15()	50
3.10.2.3	filter20()	50

3.10.2.4	filter26()	50
3.10.2.5	filter31()	50
3.11	scripts/overlays.awk File Reference	50
3.11.1	Detailed Description	51
3.11.2	Function Documentation	51
3.11.2.1	filter19()	51
3.11.2.2	filter49()	52
3.11.2.3	filter56()	52
3.11.2.4	filter63()	52
3.11.2.5	filter70()	52
3.11.2.6	filter80()	52
3.11.2.7	filter89()	53
3.11.2.8	filter104()	53
3.11.2.9	filter117()	53
3.11.2.10	filter130()	53
3.11.2.11	filter152()	53
3.12	scripts/sdcc.sh File Reference	53
3.12.1	Detailed Description	53
3.13	scripts/testver.sh File Reference	54
3.13.1	Detailed Description	54
3.14	scripts/xml.awk File Reference	54
3.14.1	Detailed Description	55
3.14.2	Function Documentation	56
3.14.2.1	filter34()	56
3.14.2.2	empty()	56
3.14.2.3	explode()	56
3.14.2.4	escape()	57
3.14.2.5	cmdSelect()	57
3.14.2.6	cmdSearch()	57
3.14.2.7	cmdSet()	58
3.14.2.8	cmdRename()	58
3.14.2.9	cmdAttrib()	58
3.14.2.10	cmdRenameAttrib()	58
3.14.2.11	cmdInsert()	59
3.14.2.12	cmdSelectInserted()	59
3.14.2.13	cmdDelete()	59
3.14.2.14	cmdDeleteAttrib()	59
3.14.2.15	cmdPrint()	60
3.14.2.16	printNode()	60
3.14.2.17	filter567()	60
3.14.2.18	filter631()	60
3.15	uVisionupdate.sh File Reference	61
3.15.1	Detailed Description	61

Chapter 1

HSK XC878 μ C Library Build Scripts

This document contains the documentation for the `scripts` folder. Scripts are written for AWK and Bourne Shell.

See also

[PDF Version](#)

1.1 Compatibility

The AWK scripts have been tested with the following interpreters:

- `awk version 20121220 (FreeBSD)`
 - Default AWK interpreter in BSD systems and OS-X
 - Also known as New AWK (NAWK)
 - This is a version of Brian Kernighan's AWK, one of the authors of *The AWK Programming Language*
- `mawk 1.3.3`
 - Default AWK interpreter in Ubuntu GNU/Linux and derivatives
 - This is Mike's AWK
- `GNU Awk 4.1.0, API: 1.0`
 - Default AWK interpreter in many GNU/Linux distributions

1.2 Layout

The `scripts` folder has the following layout:

- `scripts/`
 - Contains AWK and SH scripts for automatic build configuration, code generators and filters
- `scripts/doc/`
 - Contains the text for this document
- `scripts/templates.dbc2c/`
 - Contains the code templates for the [dbc2c.awk](#) script

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Makefile	Provides targets to build code with SDCC, generate documentation etc	5
Makefile.include	Sets some default variables that are also usable in depending projects	6
uVisionupdate.sh	Updates the Keil μ Vision configuration, with the correct include paths and overlays	61
scripts/awk2doxygen.awk	Converts an awk script containing doxygen comments into something the C parser of doxygen can handle	6
scripts/build.awk	Creates build and link instructions for a given project path	11
scripts/cstrip.awk	Separates C instructions into individual lines, streamlining the formatting for parsing in other scripts	17
scripts/dbc.sh	This script produces a make file with instructions to generate C headers from Vector DBC files	18
scripts/dbc2c.awk	This script parses Vector CAN DBs (.dbc files), such as can be created using Vector CANdb++	19
scripts/depends.awk	Creates a list of dependencies for compiling or linking	44
scripts/file2doxygen.awk	This is a doxygen filter for unsupported scripting languages	48
scripts/filter.sugar.awk	Filter certain syntactical sugar from C code	49
scripts/overlays.awk	Finds call tree manipulations for μ Vision from C files	50
scripts/sdcc.sh	Parses an sdcc config file	53
scripts/testver.sh	Implements comparison of version numbers	54
scripts/xml.awk	This script provides a small command line XML editor	54

Chapter 3

File Documentation

3.1 Makefile File Reference

Provides targets to build code with SDCC, generate documentation etc.

3.1.1 Detailed Description

Provides targets to build code with SDCC, generate documentation etc.

Target	Function
build (default)	Builds a .hex file and dependencies
all	Builds a .hex file and every .c library
dbc	Builds C headers from Vector dbc files
debug	Builds for debugging with sdcdb
printEnv	Used by scripts to determine project settings
uVision	Run uVisionupdate.sh
html	Build all html documentation under doc/
pdf	Build all pdf documentation under doc/
gh-pages	Assemble all docs for GitHub Pages publishing
clean-build	Remove build output
clean-doc	Remove doxygen output for user doc
clean	Clean everything

Override the following settings in Makefile.local if needed.

Assignment	Function
AWK	The awk interpreter
BUILDDIR	SDCC output directory
CC	Compiler
CFLAGS	Compiler flags
CPP	C preprocessor used by several scripts
CONFDIR	Location for configuration files
CANPROJDIR	Path to the CAN project

Assignment	Function
DBCDIR	Location for generated DBC headers
DOC_ALL_TARGETS	All doc/ subtargets (user, dev, dbc, scripts)
DOC_PUB_TARGETS	All gh-pages/ subtargets (user, dev, scripts)
GENDIR	Location for generated code
INCDIR	Include directory for contributed headers
OBJSUFFIX	The file name suffix for object files
HEXSUFFIX	The file name suffix for intel hex files
DATE	System date, used if <code>git</code> cannot be found
VERSION	The <code>git</code> version of the project or the date
PROJECT	The name of this project

3.2 Makefile.include File Reference

Sets some default variables that are also usable in depending projects.

3.2.1 Detailed Description

Sets some default variables that are also usable in depending projects.

The variables are document in [Makefile](#).

3.3 scripts/awk2doxygen.awk File Reference

Converts an awk script containing doxygen comments into something the C parser of doxygen can handle.

Functions

- void `filter14` ()
Initialise globals.
- void `debug` (var msg)
Prints a debugging message on stderr.
- void `filter38` ()
Fill the input buffer.
- void `initDoc` ()
Setup globals to assemble a new documentation block.
- var `genFunction` (var name)
Generates a function signature.
- void `filter84` ()
Flush the remaining input buffer.
- void `filter94` ()
Initialise file documentation.
- void `filter103` ()
Initialise documentation block.
- void `filter115` ()

- void `filter178` ()
Close documentation block.
- void `filter185` ()
Strip the indenting from the documentation.
- void `filter196` ()
Collect function parameters.
- void `filter203` ()
Detect that the function returns something.
- void `filter219` ()
Replace static regular expressions with strings.
- void `filter226` ()
Assume the current line does not contain a comment.
- void `filter241` ()
Make C comments out of awk comments.
- void `filter254` ()
Replace \$n with incol[n].
- void `filter254` ()
Append a semicolon.

3.3.1 Detailed Description

Converts an awk script containing doxygen comments into something the C parser of doxygen can handle.

Warning

This script only produces correct output for completely documented code.

3.3.2 Function Documentation

3.3.2.1 `filter14()`

```
void filter14 ( )
```

Initialise globals.

Precondition

```
BEGIN
```

3.3.2.2 `debug()`

```
void debug (
    var msg )
```

Prints a debugging message on stderr.

The debugging message is only printed if DEBUG is set.

Parameters

<i>msg</i>	The message to print
------------	----------------------

3.3.2.3 filter38()

```
void filter38 ( )
```

Fill the input buffer.

Buffering is used to inject code into previous output lines.

3.3.2.4 initDoc()

```
void initDoc ( )
```

Setup globals to assemble a new documentation block.

The following globals are reset:

- `doc(string)`: Set to enter documentation mode, contains the initial string to be able to reproduce the indentation
- `fret(bool)`: The coming function returns something
- `fargs(array)`: The coming function has the following arguments, further arguments are discarded as local variables

3.3.2.5 genFunction()

```
var genFunction (
    var name )
```

Generates a function signature.

Parameters

<i>name</i>	The function name
-------------	-------------------

Returns

A string containing a function declaration

3.3.2.6 filter84()

```
void filter84 ( )
```

Flush the remaining input buffer.

Precondition

```
END
```

3.3.2.7 filter94()

```
void filter94 ( )
```

Initialise file documentation.

Precondition

```
!doc && /^#!/
```

3.3.2.8 filter103()

```
void filter103 ( )
```

Initialise documentation block.

Precondition

```
!doc && /^[ \t]*##/
```

3.3.2.9 filter115()

```
void filter115 ( )
```

Close documentation block.

This closes a documentation block and generates a function signature for functions and filters.

Precondition

```
doc && /^[ \t]*([\^#]|$)/
```

3.3.2.10 filter178()

```
void filter178 ( )
```

Strip the indenting from the documentation.

This benefits verbatim and code formatting.

Precondition

```
doc && /^[ \t]*#/
```

3.3.2.11 filter185()

```
void filter185 ( )
```

Collect function parameters.

Precondition

```
doc && buf[line] ~ /[\\@]param(\[(in|out|in,out)\])?[ \t]/
```

3.3.2.12 filter196()

```
void filter196 ( )
```

Detect that the function returns something.

Precondition

```
doc && buf[line] ~ /[\\@](return|retval)/
```

3.3.2.13 filter203()

```
void filter203 ( )
```

Replace static regular expressions with strings.

Precondition

```
!doc && buf[line] ~ /^(("(\\.|[^\"])*"?[^\s/])*\/(\\.|[^\s/])+\/
```

3.3.2.14 filter219()

```
void filter219 ( )
```

Assume the current line does not contain a comment.

3.3.2.15 filter226()

```
void filter226 ( )
```

Make C comments out of awk comments.

Precondition

```
!doc && buf[line] ~ /^(("(\\.|[^\"])*"?[^\s#])*#
```

3.3.2.16 filter241()

```
void filter241 ( )
```

Replace \$n with incol[n].

Precondition

```
!doc && buf[line] ~ /^(("(\\.|[^"])*"?)?[^"$]*\${0-9}+)
```

3.3.2.17 filter254()

```
void filter254 ( )
```

Append a semicolon.

Precondition

```
!doc && !comment && buf[line] ~ /^[^
```

3.4 scripts/build.awk File Reference

Creates build and link instructions for a given project path.

Functions

- void [filter118](#) ()
 - This function handles all input and produces the output of the script.*
- void [verbose](#) (var name, var value)
 - Print a name/value pair if `DEBUG` is set.*
- void [verbose_values](#) (var name, var a)
 - Print a name and the values of an array if `DEBUG` is set.*
- void [verbose_keys](#) (var name, var a)
 - Print a name and the keys of an array if `DEBUG` is set.*
- var [esc](#) (var str)
 - Escape characters that are special in a regex.*
- var [compact](#) (var path)
 - Compress the given path.*
- var [flags](#) (var file, var fmt)
 - Generate context specific flags.*
- var [src_to_obj](#) (var file)
 - Convert a file name to an object file name.*
- var [src_to_origin](#) (var file)
 - Convert a file name to an origin.*
- var [obj_to_bin](#) (var file)
 - Convert an object to a binary file name.*
- var [linklist](#) (var obj)
 - Recursively get all objects required to link with the given object.*

3.4.1 Detailed Description

Creates build and link instructions for a given project path.

This script is a wrapper around `cpp -MM` and has a performance comparable to the `mkdep` tool.

The easiest way to invoke the script is by calling:

```
scripts/build.awk src/
```

This command outputs build instructions for all C and C++ files found in `src/`.

It will commonly be necessary to add custom arguments to the CPP call, i.e. because a define, an include directory or a specific version of the C/C++ standard is required:

```
scripts/build.awk src/ -I/usr/local/include -DRELEASE -std=c++14
```

These arguments have to come after the first source directory or the AWK interpreter will treat them as unknown arguments to itself instead of arguments to the script.

The script supports building from multiple sources by providing multiple input directories. In case of multiple objects with the same name the first one encountered wins.

Environment

A number of variables make it possible to steer the behaviour of `build.awk`, either by setting them in the environment or by providing them explicitly via `-vVAR=VALUE`:

Variable	Description	Default
DEBUG	Set to enable verbose output	" "
CPP	Change to pass flags to <code>cpp</code>	" <code>cpp -MM %s</code> "
FIND	Control <code>find</code> syntax	" <code>find -E %s ...</code> "
MKDIR	Command for creating build directories	" <code>mkdir -p %s</code> "
OBJSUFFIX	Set to control suffix of object files	".o"
BINSUFFIX	Set to control suffix of binaries	" "

The string "`-MM %s`" is implicitly appended to `CPP` if it set through the environment.

Special Make Variables

For better portability a set of make variables that evaluate to target local variables are defined. The variables have a default meaning and a BSD make specific fallback:

Variable	Default	BSD Fallback
<code>_IMPSRC</code>	<code>\$<</code>	<code>\${.ALLSRC: [1]}</code>
<code>_TARGET</code>	<code>\$@</code>	<code>\${.TARGET}</code>
<code>_ALLSRC</code>	<code>\$^</code>	<code>\${.ALLSRC}</code>

These can be overridden if needed.

Build Flags

There are some options to influence the build without rerunning `build.awk`:

Variable	Meaning	Default
CC	C compiler to use	OS defined
CFLAGS	Flags to the C compiler	OS defined
CXX	C++ compiler to use	OS defined
CXXFLAGS	Flags to the C++ compiler	OS defined
LDFLAGS	Linker flags	OS defined
LDLIBS	Libraries to link	OS defined
OBJDIR	Optional build output directory	undefined
COMPFLAGS*	Additional flags for compiling	undefined
LINKFLAGS*	Additional flags for linking	undefined
LINKLIBS*	Additional libraries to link	undefined

If defined the `OBJDIR` variable must terminate with a `/`.

The `COMPFLAGS*`, `LINKFLAGS*` and `LINKLIBS*` variables are variable hierarchies. E.g. for the target `foo/bar.o` the following hierarchy of compile arguments will be created:

```
 ${COMPFLAGS} ${COMPFLAGS_foo} ${COMPFLAGS_foo_bar_o}
```

Meta Targets

The output of this script provides two meta targets:

Target	Description
build	Build all binaries
all	Build everything

3.4.2 Function Documentation

3.4.2.1 filter118()

```
void filter118 ( )
```

This function handles all input and produces the output of the script.

In addition to the user-definable globals it uses the following globals to keep state:

Variable	Source	Description
SRCDIRS	Command line arguments	The list of source directories
CPPARGS	Command line arguments	Additional CPP arguments
SRCS	FIND in SRCDIRS	List of source files

Variable	Source	Description
INCLUDES	CPP output	Map: object → full sources
OBJSRC	CPP output	Map: object → source
COMPILER	CPP output	Map: object → compiler/flags
ORIGIN	CPP output	Map: origin → object

Precondition

```
BEGIN
```

3.4.2.2 verbose()

```
void verbose (
    var name,
    var value )
```

Print a name/value pair if `DEBUG` is set.

Parameters

<i>name</i>	The name to print
<i>value</i>	The value to print

3.4.2.3 verbose_values()

```
void verbose_values (
    var name,
    var a )
```

Print a name and the values of an array if `DEBUG` is set.

Note this function assumes the array is indexed in 1 increments starting at 1.

Parameters

<i>name</i>	The name to print
<i>a</i>	An array to print the values of

3.4.2.4 verbose_keys()

```
void verbose_keys (
    var name,
    var a )
```

Print a name and the keys of an array if `DEBUG` is set.

Parameters

<i>name</i>	The name to print
<i>a</i>	An array to print the keys of

3.4.2.5 esc()

```
var esc (
    var str )
```

Escape characters that are special in a regex.

Parameters

<i>str</i>	The string to escape
------------	----------------------

Returns

The escaped string

3.4.2.6 compact()

```
var compact (
    var path )
```

Compress the given path.

Eliminate `./` and `../` from the given path.

This can be broken by symlink shenanigans.

Parameters

<i>path</i>	The path to change
-------------	--------------------

Returns

The compressed path

3.4.2.7 flags()

```
var flags (
    var file,
    var fmt )
```

Generate context specific flags.

Parameters

<i>file</i>	The file to generate flags for
<i>fmt</i>	A format string to embed each flag in, e.g. " <code>{COMPFLAGS%s}</code> "

Returns

A hierarchy of flags for the given file

3.4.2.8 src_to_obj()

```
var src_to_obj (
    var file )
```

Convert a file name to an object file name.

The object suffix is taken from the OBJSUFX global.

Parameters

<i>file</i>	The file name to convert
-------------	--------------------------

Returns

The corresponding object file name

3.4.2.9 src_to_origin()

```
var src_to_origin (
    var file )
```

Convert a file name to an origin.

This can be used in conjunction with ORIGIN to determine whether there is an object for an origin. E.g. `src/foo.c` and `src/foo.h` both have the origin `src/foo` and `ORIGIN["src/foo"]` points to `foo.o`.

Parameters

<i>file</i>	The file name to convert
-------------	--------------------------

Returns

The file name stripped of its suffix

3.4.2.10 obj_to_bin()

```
var obj_to_bin (
    var file )
```

Convert an object to a binary file name.

The object and binary file name suffixes are taken from the `OBJSUFFIX` and `BINSUFFIX` globals.

Parameters

<i>file</i>	The file name to convert
-------------	--------------------------

Returns

The corresponding object file name

3.4.2.11 linklist()

```
var linklist (
    var obj )
```

Recursively get all objects required to link with the given object.

Uses `ORIGIN` and `INCLUDES` to track dependencies.

Parameters

<i>obj</i>	The object to start from
------------	--------------------------

Returns

A space separated list of object files

3.5 scripts/cstrip.awk File Reference

Separates C instructions into individual lines, streamlining the formatting for parsing in other scripts.

Functions

- void [filter18](#) ()
Initialise globals and process command line arguments.
- void [filter44](#) ()
Accumulate and preprocess files so they become easier to parse.

3.5.1 Detailed Description

Separates C instructions into individual lines, streamlining the formatting for parsing in other scripts.

Arguments starting with a dash are passed to each CPP instance, arguments the remaining arguments are treated as file names.

3.5.2 Function Documentation

3.5.2.1 filter18()

```
void filter18 ( )
```

Initialise globals and process command line arguments.

The following globals are created:

- CPP: The C preprocessor to use, either from the environment variable with the same name or the default "cpp"
- DEBUG: Defaults to the environment variable with the same name

Precondition

```
BEGIN
```

3.5.2.2 filter44()

```
void filter44 ( )
```

Accumulate and preprocess files so they become easier to parse.

Precondition

```
!visited[FILENAME]++
```

3.6 scripts/dbc.sh File Reference

This script produces a make file with instructions to generate C headers from Vector DBC files.

3.6.1 Detailed Description

This script produces a make file with instructions to generate C headers from Vector DBC files.

Parameters

*	The directories containing .dbc files
---	---------------------------------------

The script uses the `LIBPROJDIR` and `AWK` environment variables similar to the way `build.sh` uses them.

For each .dbc file a target to create an UTF-8 version of the file is created. The file is expected to use CP1252 (Windows encoding), the conversion is performed by `iconv`.

The next target uses `dbc2c.awk` to generate the desired C headers.

The meta target `dbc` generates all headers.

3.7 scripts/dbc2c.awk File Reference

This script parses Vector CAN DBs (.dbc files), such as can be created using Vector CANdb++.

Functions

- void `filter329` ()
Initialises globals.
- void `filter439` ()
Strip DOS line endings and make sure there is a new line symbol at the end of the line, so multiline definitions can be parsed.
- void `error` (var no, var msg)
Prints an error message on stderr and exits.
- void `warn` (var msg)
Prints a warning message on stderr.
- void `debug` (var msg)
Prints a debugging message on stderr.
- void `buffer` ()
Makes sure \$0 is not empty.
- var `fetchStr` ()
Special function to fetch a string from the buffer.
- var `fetch` (var types)
Fetch the next token from the input buffer, matching a given type.
- var `whole` (var re)
Returns the expression with ^ and \$ at beginning and end to make ~ match entire strings only.
- var `strip` (var str)
Remove quotes and escapes from strings.
- var `getContext` (var str)
Returns the context type for a string.
- void `getUniqueEnum` (var ret, var enum, var val, var desc)
Generates a unique name for a value table entry.
- void `fsm_discard` ()
Discards buffered symbols until an empty line is encountered.
- void `fsm_ecu` ()
Parse an ECU definition.
- void `fsm_enum` ()
Parse a value table.
- void `fsm_sig_enum` ()
Parse a value table bound to a signal.
- void `fsm_env` ()
Parse an environment variable.
- void `fsm_env_data` ()
Parse the data length count of DATA type environment variables.
- void `fsm_msg` ()
Parse a message definition.
- void `fsm_sig` ()
Parse a signal definition.
- void `fsm_comment` ()
Parse comments.
- void `fsm_attrrange` ()

- Parse a custom attribute definition.*

 - void `fsm_relatrrange` ()
- Parse a custom relation attribute definition.*

 - void `fsm_attrdefault` ()
- Parse attribute default value.*

 - var `fetch_attrval` (var attribute)

Fetches an attribute value of a given type from the read buffer.
- Parse an attribute value.*

 - void `fsm_attr` ()
- Parse a relation attribute value.*

 - void `fsm_relatrr` ()
- Parse the symbol table at the beginning of a .dbc file, bail if unsupported symbols are encountered.*

 - void `fsm_symbols` ()
- Gets a list of ECUs that transmit a certain message.*

 - void `fsm_tx` ()
- Gets a signal group.*

 - void `fsm_siggrp` ()
- Pick tokens from the input buffer and call the respective parsing functions.*

 - void `fsm_start` ()
- This starts the line wise parsing of the DBC file.*

 - void `filter1434` ()
- Returns the greatest common divider (GCD).*

 - var `euclid` (var a, var b)
- Returns a compact string representation of a rational number.*

 - var `rationalFmt` (var n, var d)
- Returns a rational string representation of a real value.*

 - var `rationalN` (var val, var base, var precision)
- Returns a rational string representation of a real value.*

 - var `rationalD` (var val, var base, var precision)
- Returns a rational string representation of a real value.*

 - var `rational` (var val, var precision)
- Applies filter chains to a given string.*

 - var `filter` (var str, var filters, var `template`)
- Populates a template line with data.*

 - var `tpl_line` (var data, var line, var `template`)
- Reads a template, substitutes place holders with data from a given array and returns it.*

 - var `template` (var data, var name)
- Set the necessary type to be able to shift something to the given bit.*

 - void `setTypes` (var array, var bitpos)
- Returns a unique signal identifier using the sigident.tpl file.*

 - var `sigident` (var sig)
- Returns a unique signal group identifier using the sigident.tpl file.*

 - var `siggrpident` (var sg)
- Generates a printable message id by removing the extended bit.*

 - var `msgjid` (var id)
- Tests a message id for the extended bit.*

 - var `msgidext` (var id)
- Print the DBC files to stdout.*

 - void `filter1841` ()

3.7.1 Detailed Description

This script parses Vector CAN DBs (.dbc files), such as can be created using Vector CANdb++.

A subset of the parsed information is output using a set of templates.

Note

Pipe the input through "iconv -f CP1252" so GNU AWK doesn't choke on non-UTF-8 characters in comments.

Warning

Templates are subject to change, which may break the output for your use case. To prevent this retain your own copy of the templates directory and set the `TEMPLATES` variable. Old templates will continue working, though they might cause deprecation warnings.

3.7.2 Environment

The script uses certain environment variables.

3.7.2.1 DEBUG

Value	Effect
0, ""	Debugging output is deactivated
1, any string != ""	Debugging output to stderr is activated
> 1	Additionally any string read is output

3.7.2.2 TEMPLATES

This variable can be used to pass the template directory to the script.

If the `LIBPROJDIR` environment variable is set it defaults to `${LIBPROJDIR}/scripts/templates.dbc2c`, otherwise it defaults to the relative path `scripts/templates.dbc2c`.

3.7.2.3 DATE

This can be used to define the date string provided to `header.tpl`.

It defaults to the output of the `date` command.

3.7.3 Value Tables

Since values in value tables only consist of a number and description, the first word of this description is used as a symbolic name for a given value.

All non-alphanumeric characters of this first word will be converted to underscores. Redundancies will be resolved by appending the value to the word that signifies the name.

This functionality is implemented in the function `getUniqueEnum()`.

3.7.4 Templates

This section describes the templates that are used by the script and the arguments passed to them. Templates are listed in the chronological order of use.

3.7.4.1 Special Attributes

Some of the arguments provided depend on custom attributes:

Template	Argument	Attribute	Object
sig.tpl	start	GenSigStartValue	Signal
msg.tpl	fast	GenMsgCycleTimeFast	Message
msg.tpl	cycle	GenMsgCycleTime	Message
msg.tpl	delay	GenMsgDelayTime	Message
msg.tpl	send	GenMsgSendType	Message
timeout.tpl	timeout	GenSigTimeoutTime	Relation (ECU to Signal)

These and more attributes are specified by the **Vector Interaction Layer**.

3.7.4.2 Inserting Data

Templates are arbitrary text files that are provided with a set of arguments. Arguments have a symbolic name through which they can be used. In the following sections they are called fields, because they are provided to the [template\(\)](#) function in an associative array.

Inserting data into a template is simple:

```
<:name:>
```

The previous example adds the data in the field `name` into the file. It can be surrounded by additional context:

```
#define <:name:> <:value:>
```

If `name` is "FOO_BAR" and `value` is 1337, this line would be resolved to:

```
#define FOO_BAR 1337
```

It may be desired to reformat some of those values. A number of special filters (see [filter\(\)](#)) as well as printf(3) style formatting is available. E.g. `name` can be converted to camel case and `value` to hex:

```
#define <:name:camel:%-16s:> <:value:%#x:>
```

The output would look like this:

```
#define fooBar          0x539
```


An important property of templates is that arguments may contain multiple lines. In that case the surrounding text is preserved for every line, which is useful to format multiline text or lists. This can be used to create lists or provide visual sugar around text:

```
+----[ <:title:%-50s:> ]----+
| <:text:%-60s:> |
+-----+
```

Output could look like this:

```
+----[ Racecar by Matt Brown ]----+
| 'Racecar - Searching for the Limit in Formula SAE' |
| is available for download from: |
| http://www.superfastmatt.com/2011/11/book.html |
+-----+
```

Multi line data is treated as an array of individual lines. Besides descriptions in DBC files multiline data can also originate from lists provided by this script in order to allow describing the relations between ECUs, messages, signals etc..

In some cases it is prudent to print lines conditionally. For that conditionals are provided:

```
<?name?>
```

If the revered field evaluates to `true`, the conditional is removed from the line. If it evaluates to `false`, the entire template line is omitted.

3.7.4.3 header.tpl

Used once with the following arguments:

Field	Type	Description
date	string	The current date
db	string[]	A list of identifiers for the parsed DBCs

3.7.4.4 file.tpl

Used for each input file with the following arguments:

Field	Type	Description
db	string	An identifier for this input file
file	string	The file name
comment	string[]	The comment text for this CANdb
ecu	string[]	A list of ECUs provided with this file

3.7.4.5 sigid.tpl

This template should only contain a single line that produces a unique identifier string for a signal, using the following arguments:

Field	Type	Description
msg	int	The message ID
msgname	string	The message name
sig	string	The signal name

Signal names are not globally unique, thus an identifier must contain a message reference to avoid name collisions.

3.7.4.6 ecu.tpl

Used for each ECU with the following arguments:

Field	Type	Description
ecu	string	An identifier for the ECU
comment	string[]	The comment text for this ECU
db	string	The input file identifier
txid	int[]	A list of message IDs belonging to messages sent by this ECU
txname	string[]	A list of message names sent by this ECU
rx	string[]	A list of signals received by this ECU
rxid	string[]	A list of unique signal identifiers received by this ECU

3.7.4.7 msg.tpl

Used for each message with the following arguments:

Field	Type	Description
msg	int	The message ID
name	string	The message name
comment	string[]	The comment text for this message
sig	string[]	A list of signal names contained in this message
sigid	string[]	A list of signal identifiers contained in this message
ecu	string	The ECU sending this message
ext	bool	Message ID is extended
dlc	int	The data length count
cycle	int	The cycle time of this message
fast	int	The fast cycle time of this message
delay	int	The minimum delay time between two transmissions
send	string	The send type (cyclic, spontaneous etc.)
sgid	string[]	A list of signal group ids
sgname	string[]	A list of signal group names

3.7.4.8 siggrp.tpl

Used for each signal group with the following arguments:

Field	Type	Description
id	string	The ID of the signal group (created using sigid.tpl)

Field	Type	Description
name	string	The name of the signal group
msg	int	The ID of the message containing this signal group
msgname	string	The name of the message containing this signal group
sig	string[]	A list of signals belonging to this signal group
sigid	string[]	A list of signal identifiers belonging to this signal group

3.7.4.9 sig.tpl

Used for each signal with the following arguments:

Field	Type	Description
name	string	The signal name
id	string	The unique signal identifier created with sigid.tpl
comment	string[]	The comment text for this signal
enum	bool	Indicates whether this signal has a value table
msg	int	The ID of the message sending this signal
sgid	string[]	The signal groups containing this signal
sgname	string[]	The names of the signal groups containing this signal
ecu	string[]	A list of the ECUs receiving this signal
intel	bool	Intel (little endian) style signal
motorola	bool	Motorola (big endian) style signal
signed	bool	The signal is signed
sbit	int	The start bit (meaning depends on endianness)
len	int	The signal length
start	int	The initial (default) signal value (raw)
calc16	string[]	A rational conversion function (see calc16)
min	int	The raw minimum value
max	int	The raw maximum value
off	int	The raw offset value
getbuf	string[]	The output of sig_getbuf.tpl
setbuf	string[]	The output of sig_setbuf.tpl

3.7.4.9.1 calc16

A rational conversion function for the raw signal value x and formatting factor fmt into a real value as defined by the linear factor and offset in the DBC, this function uses up to 16bit integers.

3.7.4.9.2 sig_getbuf.tpl, sig_setbuf.tpl

These templates can be used to construct static byte wise signal getters and setters.

For signed signals `sig_getbuf.tpl` is first called with the following arguments:

Field	Type	Description
sign	string	"-"
byte	int	The byte containing the most significant bit
align	int	The position of the most significant bit in the byte

Field	Type	Description
msk	int	1
pos	int	The position in front of the entire read signal
int8	bool	Indicates whether an 8 bit integer suffices to contain the signal
int16	bool	Indicates whether a 16 bit integer suffices to contain the signal
int32	bool	Indicates whether a 32 bit integer suffices to contain the signal

These arguments can be used to duplicate the signed bit and shift it in front.

Both templates are used for each touched signal byte with the following arguments:

Field	Type	Description
sign	string	"+"
byte	int	The signal byte
align	int	The least significant bit within the byte belonging to the signal
msk	int	A bit mask to mask the aligned signal bits
pos	int	The position to shift the resulting bits to
int8	bool	Indicates whether an 8 bit integer suffices to address the desired bit
int16	bool	Indicates whether a 16 bit integer suffices to address the desired bit
int32	bool	Indicates whether a 32 bit integer suffices to address the desired bit

3.7.4.9.3 sig_enum.tpl, sig_enumval.tpl

In case a value table is assigned to the signal, `sig_enum.tpl` is called with all the arguments provided to `sig.tpl`.

For each entry in the value table `sig_enumval.tpl` is called with these additional arguments:

Field	Type	Description
enumval	int	The value
enumname	string	The name of the value
comment	string[]	The comment part of the value description

3.7.4.10 timeout.tpl

Used for each timeout with the following arguments:

Field	Type	Description
ecu	string	The ECU that times out
sig	string	The signal that is expected by the ECU
sigid	string	The unique identifier for the expected signal
timeout	int	The timeout time
msg	int	The ID of the CAN message containing the signal
msgname	string	The name of the CAN message containing the signal

3.7.4.11 enum.tpl

Invoked for every value table with the following arguments:

Field	Type	Description
enum	string	The name of the value table
db	string	The name of the CAN DB this enum was defined in

3.7.4.11.1 enumval.tpl

Invoked for every value defined in a value table. All the template arguments for `enum.tpl` are available in addition to the following arguments:

Field	Type	Description
val	int	The value
name	string	The symbolic name for the value
comment	string[]	The comment part of the value description

3.7.5 Function Documentation

3.7.5.1 filter329()

```
void filter329 ( )
```

Initialises globals.

Precondition

```
BEGIN
```

3.7.5.2 filter439()

```
void filter439 ( )
```

Strip DOS line endings and make sure there is a new line symbol at the end of the line, so multiline definitions can be parsed.

3.7.5.3 error()

```
void error (
    var no,
    var msg )
```

Prints an error message on stderr and exits.

Parameters

<i>no</i>	The number to set errno to
<i>msg</i>	The error message

3.7.5.4 warn()

```
void warn (
    var msg )
```

Prints a warning message on stderr.

Parameters

<i>msg</i>	The message to print
------------	----------------------

3.7.5.5 debug()

```
void debug (
    var msg )
```

Prints a debugging message on stderr.

The debugging message is only printed if DEBUG is set.

Parameters

<i>msg</i>	The message to print
------------	----------------------

3.7.5.6 buffer()

```
void buffer ( )
```

Makes sure \$0 is not empty.

3.7.5.7 fetchStr()

```
var fetchStr ( )
```

Special function to fetch a string from the buffer.

This is a special case, because strings may span multiple lines. This function supports strings with up to 256 lines.

Returns

The fetched string

3.7.5.8 fetch()

```
var fetch (
    var types )
```

Fetch the next token from the input buffer, matching a given type.

Parameters

<i>types</i>	A regular expression describing the type of data to be fetched
--------------	--

Returns

The fetched string of data

3.7.5.9 whole()

```
var whole (
    var re )
```

Returns the expression with `^` and `$` at beginning and end to make `~` match entire strings only.

Parameters

<i>re</i>	The expression to wrap
-----------	------------------------

Returns

An expression for matching entire strings

3.7.5.10 strip()

```
var strip (
    var str )
```

Remove quotes and escapes from strings.

This function is used by [fetchStr\(\)](#).

Parameters

<i>str</i>	The string to unescape
------------	------------------------

Returns

The literal string

3.7.5.11 getContext()

```
var getContext (
    var str )
```

Returns the context type for a string.

Parameters

<i>str</i>	The string to interpret
------------	-------------------------

Return values

<i>sig</i>	The context is a signal
<i>msg</i>	The context is a message
<i>ecu</i>	The context is an ECU
<i>env</i>	The context is an environment variable
<i>db</i>	The context is the DB

3.7.5.12 getUniqueEnum()

```
void getUniqueEnum (
    var ret,
    var enum,
    var val,
    var desc )
```

Generates a unique name for a value table entry.

Updates:

- `obj_enum_count[enum, name] = (int)`

Sets the following fields in the given array:

- `name`: A unique identifier
- `desc`: The description
- `invalid`: No valid identifier was in the description (bool)
- `duplicate`: The identifier was already in use (bool)

Parameters

<i>ret</i>	An array to return the data set in
<i>enum</i>	The identifier of the value table
<i>val</i>	The value
<i>desc</i>	The description string to fetch a name from

3.7.5.13 fsm_discard()

```
void fsm_discard ( )
```

Discards buffered symbols until an empty line is encountered.

This is used to skip the list of supported symbols at the beginning of a dbc file.

3.7.5.14 fsm_ecu()

```
void fsm_ecu ( )
```

Parse an ECU definition.

Token: BU_

Creates:

- * ind_ecu[cnt_ecu++] = ecu
- * obj_ecu[ecu]
- * obj_ecu_db[ecu] = FILENAME
- * obj_db_ecu[FILENAME, p] = ecu

3.7.5.15 fsm_enum()

```
void fsm_enum ( )
```

Parse a value table.

Token: VAL_TABLE_

Creates:

- 1 ind_enum[cnt_enum++] = enum
- 1 obj_enum_db[enum] = FILENAME
- * obj_enum_val[enum, i] = val
- * obj_enum_name[enum, i] = name
- * obj_enum_desc[enum, i] = desc
- * obj_enum_invalid[enum, i] = (bool)
- * obj_enum_duplicate[enum, i] = (bool)

3.7.5.16 fsm_sig_enum()

```
void fsm_sig_enum ( )
```

Parse a value table bound to a signal.

Token: VAL_

Creates:

- 1 obj_sig_enum[msgid, sig]
- * obj_sig_enum_val[msgid, sig, i] = val
- * obj_sig_enum_name[msgid, sig, i] = name
- * obj_sig_enum_desc[enum, i] = desc
- * obj_sig_enum_invalid[enum, i] = (bool)
- * obj_sig_enum_duplicate[enum, i] = (bool)

3.7.5.17 fsm_env()

```
void fsm_env ( )
```

Parse an environment variable.

Token: EV_

Creates:

- 1 ind_env[cnt_env++] = name
- 1 obj_env[name] = val
- 1 obj_env_type[name] = ("INT"|"FLOAT"|"DATA")
- 1 obj_env_min[name] = (float)
- 1 obj_env_max[name] = (float)
- 1 obj_env_unit[name] = (string)

3.7.5.18 fsm_env_data()

```
void fsm_env_data ( )
```

Parse the data length count of DATA type environment variables.

Token: ENVVAR_DATA_

Creates:

- 1 obj_env_dlc[name] = (int)

3.7.5.19 fsm_msg()

```
void fsm_msg ( )
```

Parse a message definition.

Token: BO_

Creates:

- 1 ind_msg[cnt_msg++] = id
- 1 obj_msg_name[id] = name
- 1 obj_msg_dlc[id] = dlc
- 1 obj_msg_tx[id] = ecu
- 1 obj_ecu_tx[ecu, i] = id

3.7.5.20 fsm_sig()

```
void fsm_sig ( )
```

Parse a signal definition.

Token: SG_

Creates:

- 1 ind_sig[cnt_sig++] = msgid, name
- 1 obj_sig_name[msgid, name] = name
- 1 obj_sig_msgid[msgid, name] = msgid
- 1 obj_sig_multiplexor[msgid, name] = (bool)
- 1 obj_sig_multiplexed[msgid, name] = (int)
- 1 obj_sig_sbit[msgid, name] = (uint)
- 1 obj_sig_len[msgid, name] = (uint)
- 1 obj_sig_intel[msgid, name] = (bool)
- 1 obj_sig_signed[msgid, name] = (bool)
- 1 obj_sig_fac[msgid, name] = (float)
- 1 obj_sig_off[msgid, name] = (float)
- 1 obj_sig_min[msgid, name] = (float)
- 1 obj_sig_max[msgid, name] = (float)
- 1 obj_sig_unit[msgid, name] = (string)
- * obj_sig_rx[msgid, name, i] = ecu
- * obj_ecu_rx[ecu, p] = msgid, name
- * obj_msg_sig[msgid, p] = msgid, name

3.7.5.21 fsm_comment()

```
void fsm_comment ( )
```

Parse comments.

Token: CM_

Creates one of:

- 1 obj_db_comment[FILENAME]
- 1 obj_ecu_comment[name]
- 1 obj_env_comment[name]
- 1 obj_msg_comment[msgid]
- 1 obj_sig_comment[msgid, name]

3.7.5.22 fsm_atrrange()

```
void fsm_atrrange ( )
```

Parse a custom attribute definition.

Token: BA_DEF_

Creates:

- 1 ind_attr[cnt_attr++] = name
- 1 obj_attr_context[name] = ("sig"|"msg"|"ecu"|"env"|"db")
- 1 obj_attr_type[name] = ("INT"|"ENUM"|"STRING")
- ? obj_attr_min[name] = (float)
- ? obj_attr_max[name] = (float)
- * obj_attr_enum[name, i] = (string)
- ? obj_attr_str[name] = (string)

3.7.5.23 fsm_relattrrange()

```
void fsm_relattrrange ( )
```

Parse a custom relation attribute definition.

Token: BA_DEF_REL_

Creates:

- 1 ind_attr[cnt_attr++] = name
- 1 obj_attr_context[name] = "rel"
- 1 obj_attr_from[name] = ("sig"|"msg"|"ecu"|"env"|"db")
- 1 obj_attr_to[name] = ("sig"|"msg"|"ecu"|"env"|"db")
- 1 obj_attr_type[name] = ("INT"|"ENUM"|"STRING")
- ? obj_attr_min[name] = (float)
- ? obj_attr_max[name] = (float)
- * obj_attr_enum[name, i] = (string)
- ? obj_attr_str[name] = (string)

3.7.5.24 fsm_attrdefault()

```
void fsm_attrdefault ( )
```

Parse attribute default value.

Token: BA_DEF_DEF_

Creates:

- 1 obj_attr_default[name] = value
- * obj_msg_attr[msgid, name]
- * obj_sig_attr[msgid, signame, name]
- * obj_db_attr[FILENAME, name]

3.7.5.25 fetch_attrval()

```
var fetch_attrval (
    var attribute )
```

Fetches an attribute value of a given type from the read buffer.

Parameters

<i>attribute</i>	The attribute type identifier
------------------	-------------------------------

Returns

The value of the chosen type

3.7.5.26 fsm_attr()

```
void fsm_attr ( )
```

Parse an attribute value.

Token: BA_

Creates one of:

- 1 obj_sig_attr[msgid, signame, name] = value
- 1 obj_msg_attr[msgid, name] = value
- 1 obj_ecu_attr[ecu, name] = value
- 1 obj_db_attr[FILENAME, name] = value

3.7.5.27 fsm_relattr()

```
void fsm_relattr ( )
```

Parse a relation attribute value.

Token: BA_REL_

Creates:

- 1 ind_rel_attr[cnt_rel_attr++] = name, from, to
- 1 obj_rel_attr[name, from, to] = value
- 1 obj_rel_attr_name[name, from, to] = name
- 1 obj_rel_attr_from[name, from, to] = from
- 1 obj_rel_attr_to[name, from, to] = to

The types of to and from are recorded in:

- obj_attr_from[name]
- obj_attr_to[name]

3.7.5.28 fsm_symbols()

```
void fsm_symbols ( )
```

Parse the symbol table at the beginning of a .dbc file, bail if unsupported symbols are encountered.

Token: NS_

3.7.5.29 fsm_tx()

```
void fsm_tx ( )
```

Gets a list of ECUs that transmit a certain message.

This may appear when several device options are available.

Token: BO_TX_BU_

Creates:

- 1 obj_ecu_tx[ecu, i] = msgid

3.7.5.30 fsm_siggrp()

```
void fsm_siggrp ( )
```

Gets a signal group.

Token: SIG_GROUP_

Creates:

- 1 ind_siggrp[cnt_siggrp++] = msgid, name
- 1 obj_siggrp[msgid, name] = name
- 1 obj_siggrp_msg[msgid, name] = msgid
- * obj_siggrp_sig[msgid, name, i] = sig
- * obj_sig_grp[msgid, sig, p] = msgid, name
- * obj_msg_grp[msgid, p] = msgid, name

3.7.5.31 fsm_start()

```
void fsm_start ( )
```

Pick tokens from the input buffer and call the respective parsing functions.

Creates:

- 1 ind_db[cnt_db++] = FILENAME
- 1 obj_db[FILENAME]

3.7.5.32 filter1434()

```
void filter1434 ( )
```

This starts the line wise parsing of the DBC file.

3.7.5.33 euclid()

```
var euclid (
    var a,
    var b )
```

Returns the greatest common divider (GCD).

Parameters

<i>a</i>	An integer
<i>b</i>	An integer

Returns

The greatest common divider of a and b

3.7.5.34 rationalFmt()

```
var rationalFmt (
    var n,
    var d )
```

Returns a compact string representation of a rational number.

Parameters

<i>n</i>	The numerator
<i>d</i>	The denominator

Returns

The given rational number as a string

3.7.5.35 rationalN()

```
var rationalN (
    var val,
    var base,
    var precision )
```

Returns a rational string representation of a real value.

This function builds the value around the numerator.

Parameters

<i>val</i>	The real value to return as a rational
<i>base</i>	The logical number base to generate the rational from
<i>precision</i>	The maximum number of bits for either rational component

Returns

A rational string representation of the given value

3.7.5.36 rationalID()

```
var rationalD (
    var val,
    var base,
    var precision )
```


Returns a rational string representation of a real value.

This function builds the value around the denominator.

Parameters

<i>val</i>	The real value to return as a rational
<i>base</i>	The logical number base to generate the rational from
<i>precision</i>	The maximum number of bits for either rational component

Returns

A rational string representation of the given value

3.7.5.37 rational()

```
var rational (
    var val,
    var precision )
```

Returns a rational string representation of a real value.

This uses the different rational*() functions to find a minimal representation of the value.

Parameters

<i>val</i>	The real value to return as a rational
<i>precision</i>	The maximum number of bits for either rational component

Returns

A rational string representation of the given value

3.7.5.38 filter()

```
var filter (
    var str,
    var filters,
    var template )
```

Applies filter chains to a given string.

Filters are a colon separated lists of the following filter commands:

Command	Effect
low	Convert to lower case
up	Convert to upper case
camel	Convert to camel case
uncamel	Convert camel case to _ separated
%...	A printf(3) style format specification

Parameters

<i>str</i>	The string to apply the filters to
<i>filters</i>	The list of filters
<i>template</i>	The name of the current template

Returns

The converted string

3.7.5.39 tpl_line()

```
var tpl_line (
    var data,
    var line,
    var template )
```

Populates a template line with data.

Multiline data in a template needs to be in its own line.

Lines with empty data fields are removed.

Identifiers in templates have the following shape: "<:" name ">:"

Additionally boolean filters can be installed: "<?" name ">?"

If the variable addressed in the filter evaluates to true, the filter is removed, otherwise the entire line is removed.

Parameters

<i>data</i>	The array containing field data
<i>line</i>	The line to perform substitutions in
<i>template</i>	The name of the template this line comes from, this is used to warn about deprecated arguments

Returns

The line(s) with performed substitutions

3.7.5.40 template()

```
var template (
    var data,
    var name )
```

Reads a template, substitutes place holders with data from a given array and returns it.

Parameters

<i>data</i>	The array to take data from
<i>name</i>	The name of the template file

Returns

The filled up template

3.7.5.41 setTypes()

```
void setTypes (
    var array,
    var bitpos )
```

Set the necessary type to be able to shift something to the given bit.

Creates the entries int8, int16 and int32 in the given arrays, with the fitting type set to the value 1 and the others to 0.

Parameters

<i>array</i>	The array put the entries into
<i>bitpos</i>	The bit that needs to be addressable

3.7.5.42 sigident()

```
var sigident (
    var sig )
```

Returns a unique signal identifier using the sigident.tpl file.

Returns an empty string if the template is missing.

Parameters

<i>sig</i>	The signal reference consisting of message and signal name
------------	--

Returns

A unique signal identifier

3.7.5.43 siggrpident()

```
var siggrpident (
    var sg )
```

Returns a unique signal group identifier using the sigident.tpl file.

Returns an empty string if the template is missing.

Parameters

<i>sg</i>	The signal group reference consisting of message and signal name
-----------	--

Returns

A unique signal identifier

3.7.5.44 msgid()

```
var msgid (
    var id )
```

Generates a printable message id by removing the extended bit.

Parameters

<i>id</i>	The message id to return
-----------	--------------------------

Returns

The message id without the extended bit

3.7.5.45 msgidext()

```
var msgidext (
    var id )
```

Tests a message id for the extended bit.

Parameters

<i>id</i>	The message id to check
-----------	-------------------------

Return values

<i>1</i>	The message is extended
<i>0</i>	The message is not extended

3.7.5.46 filter1841()

```
void filter1841 ( )
```

Print the DBC files to stdout.

Precondition

END

3.8 scripts/depends.awk File Reference

Creates a list of dependencies for compiling or linking.

Functions

- var `testf` (var file)
Tests whether a file can be opened.
- var `rescape` (var str)
Escape the given string for literal use in a regular expression.
- var `sescape` (var str)
Escape the given string for command line use.
- var `extract` (var a)
Returns an arbitrary index from an array and deletes it.
- var `any` (var a)
Returns whether any of the entries in a given array evaluate to true.
- var `compact` (var path)
Returns a compacted version of the path.
- void `filter184` ()
Perform recursive include and output C/C++ file names.

3.8.1 Detailed Description

Creates a list of dependencies for compiling or linking.

The command line arguments are used to produce a CPP command. The given files need to be processed by CPP in order to make sure macros and conditionals are (correctly) expanded.

The output is filtered to only print files in the same directory or subdirectories of the given file. Paths can be added explicitly by using the `-I` argument.

The following arguments receive special treatment:

Argument	Description
<code>-compile</code>	Choose to produce a dependency list for compiling
<code>-link</code>	Choose to produce a dependency list for linking
<code>-I<path></code>	Paths are added to the output filter

3.8.2 Modes of Operation

The script can either create a dependency list for compiling or linking.

In any mode the given file (multiple files can be chosen as well, but there is no useful use case) is passed to the CPP. The CPP resolves all includes.

The includes are filtered from the CPP output by this script.

In compile mode all files and includes are output once. In link mode includes are instead checked for heaving a corresponding C/C++ file that ends with the suffix `SUFX`. Only in that case is the file name printed and also recursively passed to the CPP.

Modes can be combined.

3.8.3 Environment

If the following arguments are not set using AWK's `-v` argument, they can be set as environment variables.

Variable	Description
DEBUG	If set output debugging information on stderr
CPP	The C/C++ preprocessor command, defaults to <code>cpp</code>
SUFX	The file name suffix for c/c++ files

3.8.3.1 SUFX

The SUFX variable defaults to the file ending of the first file name given in the arguments.

Non-existing files are ignored during list-creation, this can be used to set the file ending by providing the desired file ending as the first argument:

```
awk -f depends.awk .c -link <file>
```

3.8.4 Function Documentation

3.8.4.1 testf()

```
var testf (
    var file )
```

Tests whether a file can be opened.

Parameters

<i>file</i>	The name of the file to test
-------------	------------------------------

Return values

1	The file exists and can be read
0	The file cannot be opened

3.8.4.2 rescape()

```
var rescape (
```

```
var str )
```

Escape the given string for literal use in a regular expression.

Parameters

<i>str</i>	The string to escape
------------	----------------------

Returns

The escaped string

3.8.4.3 sescape()

```
var sescape (  
    var str )
```

Escape the given string for command line use.

Parameters

<i>str</i>	The string to escape
------------	----------------------

Returns

The escaped string

3.8.4.4 extract()

```
var extract (  
    var a )
```

Returns an arbitrary index from an array and deletes it.

Parameters

<i>a</i>	The array to fetch an arbitrary index from
----------	--

Returns

An array index or nothing, if the array is empty

3.8.4.5 any()

```
var any (  
    var a )
```

Returns whether any of the entries in a given array evaluate to true.

Parameters

<i>a</i>	The array to check for true entries
----------	-------------------------------------

Return values

<i>0</i>	No true entries
<i>1</i>	At least one entry evaluates to true

3.8.4.6 compact()

```
var compact (
    var path )
```

Returns a compacted version of the path.

This gets rid of ../ by removing the previous path.

Parameters

<i>path</i>	The path to compact
-------------	---------------------

Returns

The compacted path

3.8.4.7 filter184()

```
void filter184 ( )
```

Perform recursive include and output C/C++ file names.

- Setup environment settable globals
- Initialize escape tables for the [rescape\(\)](#) and [sescape\(\)](#) functions
- Read command line arguments
 - Assemble the CPP command
 - Collect files to pass to cpp
 - Guess the project paths from the given files
 - Detect SUFX using the first file encountered
- Process files recursively

Precondition

BEGIN

3.9 scripts/file2doxygen.awk File Reference

This is a doxygen filter for unsupported scripting languages.

Functions

- void `filter13` ()
Initialise file documentation.
- void `filter22` ()
Print file documentation.
- void `filter31` ()
Mark the end of the documentation block.

3.9.1 Detailed Description

This is a doxygen filter for unsupported scripting languages.

It's pupose is to at least produce file documentation for those languages, as such it simply provides the file documentation at the beginning of a script (the comment after the shebang).

3.9.2 Function Documentation

3.9.2.1 `filter13()`

```
void filter13 ( )
```

Initialise file documentation.

Precondition

```
!doc && /^#!//
```

3.9.2.2 `filter22()`

```
void filter22 ( )
```

Print file documentation.

Precondition

```
doc && /^#/
```

3.9.2.3 filter31()

```
void filter31 ( )
```

Mark the end of the documentation block.

Precondition

```
doc
```

3.10 scripts/filter.sugar.awk File Reference

Filter certain syntactical sugar from C code.

Functions

- void [filter10](#) ()
Remove indented preprocessor instructions, they are usually just in place hacks that don't need to show up in the docs.
- void [filter15](#) ()
Detect the beginning of a documentation block.
- void [filter20](#) ()
Detect the end of a documentation block.
- void [filter26](#) ()
Align documentation so verbatim and code sections are formatted correctly.
- void [filter31](#) ()
Print the updated line.

3.10.1 Detailed Description

Filter certain syntactical sugar from C code.

3.10.2 Function Documentation

3.10.2.1 filter10()

```
void filter10 ( )
```

Remove indented preprocessor instructions, they are usually just in place hacks that don't need to show up in the docs.

Precondition

```
 /^[ \t]+#/
```

3.10.2.2 filter15()

```
void filter15 ( )
```

Detect the beginning of a documentation block.

Precondition

```
/\/*\*/
```

3.10.2.3 filter20()

```
void filter20 ( )
```

Detect the end of a documentation block.

Precondition

```
/*\*/
```

3.10.2.4 filter26()

```
void filter26 ( )
```

Align documentation so verbatim and code sections are formatted correctly.

Precondition

```
comment
```

3.10.2.5 filter31()

```
void filter31 ( )
```

Print the updated line.

Precondition

```
1
```

3.11 scripts/overlays.awk File Reference

Finds call tree manipulations for μ Vision from C files.

Functions

- void `filter19` ()
Pass all arguments to `cstrip.awk` and pass the output to `TMPFILE`.
- void `filter49` ()
Reduce nesting depth.
- void `filter56` ()
Just for debugging level > 1, print the current input line.
- void `filter63` ()
Increase nesting depth.
- void `filter70` ()
Get filename, useful for debugging.
- void `filter80` ()
The `hsk_isr_rootN()` function is present, so an ISR call tree can be built.
- void `filter89` ()
Gather interrupts.
- void `filter104` ()
Catch shared ISRs.
- void `filter117` ()
Catch timer0/timer1 ISRs.
- void `filter130` ()
Catch external interrupts.
- void `filter152` ()
Remove `TMPFILE` and print assembled data.

3.11.1 Detailed Description

Finds call tree manipulations for μ Vision from C files.

This script directly makes use of the coding conventions of the `hsk_libs` and uses internal knowledge, which makes it useless for any other purpose.

3.11.2 Function Documentation

3.11.2.1 `filter19()`

```
void filter19 ( )
```

Pass all arguments to `cstrip.awk` and pass the output to `TMPFILE`.

Creates the following globals:

- `DEBUG`: Created from the environment variable with the same name
- `LIBPROJDIR`: Created from the environment variable with the same name it is used to access `cstrip.awk`
- `TMPFILE`: The file containing the output of `cstrip.awk`

Precondition

```
BEGIN
```

3.11.2.2 filter49()

```
void filter49 ( )
```

Reduce nesting depth.

Precondition

```
/\}/
```

3.11.2.3 filter56()

```
void filter56 ( )
```

Just for debugging level > 1, print the current input line.

Precondition

```
DEBUG > 1
```

3.11.2.4 filter63()

```
void filter63 ( )
```

Increase nesting depth.

Precondition

```
/\
```

3.11.2.5 filter70()

```
void filter70 ( )
```

Get filename, useful for debugging.

Precondition

```
/^#[0-9]+".*"/
```

3.11.2.6 filter80()

```
void filter80 ( )
```

The hsk_isr_rootN() function is present, so an ISR call tree can be built.

Precondition

```
/^void hsk_isr_root[0-9]+\(.*\) (___)?using [0-9]+$/
```

3.11.2.7 filter89()

```
void filter89 ( )
```

Gather interrupts.

Precondition

```
/^void .*\(void\) (\_\_)?interrupt [0-9]+ (\_\_)?using [0-9]+$/
```

3.11.2.8 filter104()

```
void filter104 ( )
```

Catch shared ISRs.

Precondition

```
/^hsk_isr[0-9]+\.[a-zA-Z0-9_]+=&[a-zA-Z0-9_]+;/
```

3.11.2.9 filter117()

```
void filter117 ( )
```

Catch timer0/timer1 ISRs.

Precondition

```
/^hsk_timer[0-9]+_setup\(.*, \&[a-zA-Z0-9_]+\);/
```

3.11.2.10 filter130()

```
void filter130 ( )
```

Catch external interrupts.

Precondition

```
/^hsk_ex_channel_enable\([a-zA-Z0-9]+,.*, \&[a-zA-Z0-9_]+\);/
```

3.11.2.11 filter152()

```
void filter152 ( )
```

Remove TMPFILE and print assembled data.

Precondition

```
END
```

3.12 scripts/sdcc.sh File Reference

Parses an sdcc config file.

3.12.1 Detailed Description

Parses an sdcc config file.

Parameters

*	All arguments are treated as config files
---	---

Expects `CC` in the environment. If `CC` does not refer to a version of SDCC, the script terminates with empty output.

Configuration files contain make code and have sections, the first section is unconditional and thus always printed. The following sections are opened with a condition. Conditions stand in a single line using the following syntax:

```
"[" condition "]"
```

The string "SDCC" within the condition is replaced with the version of SDCC. The condition is then passed to the [testver.sh](#) script.

3.13 scripts/testver.sh File Reference

Implements comparison of version numbers.

3.13.1 Detailed Description

Implements comparison of version numbers.

This works by splitting the digits at the ".". Any operator accepted by `test(1)` works, as long as the operator is applicable to the digit. Missing digits (present in the other operand) are assumed 0.

Parameters

1	First version number
2	Comparison operator
3	Second version number

3.14 scripts/xml.awk File Reference

This script provides a small command line XML editor.

Functions

- void [filter34](#) ()
Parse arguments and initialise globals.
- var [empty](#) (var array)
Return whether an array is empty.
- var [explode](#) (var str, var results)
Split a string containing attributes into a string array with "attribute=value" entries.
- var [escape](#) (var str)

- Escapes quotation marks and backslashes with backslashes.*

 - void `cmdSelect` (var str)

This function lets you define a selection.
 - void `cmdSearch` (var str)

This function selects any subtree of the current selection that matches the given selection filter.
 - void `cmdSet` (var value)

Changes the content of a node.
 - void `cmdRename` (var name)

Changes the tag name of a node.
 - void `cmdAttrib` (var str)

Changes an attribute of a node.
 - void `cmdRenameAttrib` (var str)

Changes the name of an attribute.
 - void `cmdInsert` (var str)

Inserts new nodes into all selected nodes, uses the same syntax as `cmdSelect()` does.
 - void `cmdSelectInserted` ()

Select the nodes created during the last insert operation.
 - void `cmdDelete` ()

Unhooks a selected node from the tree, it's still there and can be navigated out of by selecting "..".
 - void `cmdDeleteAttrib` (var name)

Deletes a named attribute.
 - void `cmdPrint` ()

Print the current selection.
 - void `printNode` (var indent, var node)

Prints children and contents of the given node.
 - void `filter567` ()

Parse the XML tree.
 - void `filter631` ()

Execute the specified commands.

3.14.1 Detailed Description

This script provides a small command line XML editor.

It parses the subset of XML used by ARM Keil μ Vision configuration files and provides arguments to navigate, search, edit and print the parsed XML tree.

Every command applies to the current selection. The current selection may refer to several nodes in the tree.

The command syntax is: "-" command [":" argument]

The following command line arguments are supported:

Command	Function	Description
select	<code>cmdSelect()</code>	Selects a path using a filter argument
search	<code>cmdSearch()</code>	Selects all the subtrees matching the given filter argument
set	<code>cmdSet()</code>	Sets the data of a node
rename	<code>cmdRename()</code>	Changes the tag name of a node
attrib	<code>cmdAttrib()</code>	Sets a named attribute
renameAttrib	<code>cmdRenameAttrib()</code>	Renames an attribute
insert	<code>cmdInsert()</code>	Inserts a new child node into the selected nodes
selectInserted	<code>cmdSelectInserted()</code>	Select all nodes created during the last insert operation
delete	<code>cmdDelete()</code>	Removes the selected nodes from the tree
deleteAttrib	<code>cmdDeleteAttrib()</code>	Removes an attribute from the selected nodes
print	<code>cmdPrint()</code>	Print the children and data of the selected nodes

3.14.2 Function Documentation

3.14.2.1 filter34()

```
void filter34 ( )
```

Parse arguments and initialise globals.

Precondition

```
BEGIN
```

3.14.2.2 empty()

```
var empty (
    var array )
```

Return whether an array is empty.

Parameters

<i>array</i>	The array to check
--------------	--------------------

Return values

1	The array is empty
0	The array contains at least one element

3.14.2.3 explode()

```
var explode (
    var str,
    var results )
```

Split a string containing attributes into a string array with "attribute=value" entries.

Parameters

<i>str</i>	The string to split into attributes
<i>results</i>	The array to store the results in

Returns

The count of attributes

3.14.2.4 `escape()`

```
var escape (
    var str )
```

Escapes quotation marks and backslashes with backslashes.

Parameters

<i>str</i>	The string to escape
------------	----------------------

Returns

The escaped string

3.14.2.5 `cmdSelect()`

```
void cmdSelect (
    var str )
```

This function lets you define a selection.

A selection filter is a series of node definitions divided by /. Identifiers may contain glob patterns.

A / at the beginning of the filter selects the root node, which contains the root nodes of all XML trees parsed. Otherwise the filter is relative to the current selections.

The node ./ refers to the current node and ../ to the parent node. This can be used to move through the tree relative to the current selection or to select the parent of a node that matches a filtering condition.

A node selection has the following syntax: node = tag ["[" attributes "]"] ["=" value]

Attributes have the following syntax: attributes = attribute "=" (value | "" value "") [" " attributes]

Values are strings or glob patterns.

Parameters

<i>str</i>	The selection filter
------------	----------------------

3.14.2.6 `cmdSearch()`

```
void cmdSearch (
    var str )
```

This function selects any subtree of the current selection that matches the given selection filter.

The filter syntax is identical with that of [cmdSelect\(\)](#).

Parameters

<i>str</i>	The selection filter
------------	----------------------

3.14.2.7 cmdSet()

```
void cmdSet (
    var value )
```

Changes the content of a node.

This does not affect subnodes.

Parameters

<i>value</i>	The value to set the node content to
--------------	--------------------------------------

3.14.2.8 cmdRename()

```
void cmdRename (
    var name )
```

Changes the tag name of a node.

Parameters

<i>name</i>	The new tag name
-------------	------------------

3.14.2.9 cmdAttrib()

```
void cmdAttrib (
    var str )
```

Changes an attribute of a node.

It accepts a single string in the shape: attribute "=" value

Parameters

<i>str</i>	A single attribute definition
------------	-------------------------------

3.14.2.10 cmdRenameAttrib()

```
void cmdRenameAttrib (
    var str )
```

Changes the name of an attribute.

If the original attribute does not exist the new one will be added.

If an attribute with the new name already exists it will be overwritten.

It accepts a single string in the shape: oldname "=" newname

Parameters

<i>str</i>	A single attribute renaming instruction
------------	---

3.14.2.11 cmdInsert()

```
void cmdInsert (
    var str )
```

Inserts new nodes into all selected nodes, uses the same syntax as [cmdSelect\(\)](#) does.

Parameters

<i>str</i>	A node definition like the ones used for selection filters
------------	--

3.14.2.12 cmdSelectInserted()

```
void cmdSelectInserted ( )
```

Select the nodes created during the last insert operation.

3.14.2.13 cmdDelete()

```
void cmdDelete ( )
```

Unhooks a selected node from the tree, it's still there and can be navigated out of by selecting "..".

3.14.2.14 cmdDeleteAttrib()

```
void cmdDeleteAttrib (
    var name )
```

Deletes a named attribute.

Parameters

<i>name</i>	The name of the attribute to remove
-------------	-------------------------------------

3.14.2.15 cmdPrint()

```
void cmdPrint ( )
```

Print the current selection.

3.14.2.16 printNode()

```
void printNode (
    var indent,
    var node )
```

Prints children and contents of the given node.

Parameters

<i>indent</i>	The indentation depth of the current node
<i>node</i>	The node to print

3.14.2.17 filter567()

```
void filter567 ( )
```

Parse the XML tree.

Abbreviations:

- d = depth
- c = count

Properties:

- tags [d, c]
- contents [d, c]
- attributeNames [d, c, i]
- attributeValues [d, c, i]
- children [d, c, i]
- parent [d, c]

3.14.2.18 filter631()

```
void filter631 ( )
```

Execute the specified commands.

Precondition

END

3.15 uVisionupdate.sh File Reference

Updates the Keil μ Vision configuration, with the correct include paths and overlays.

3.15.1 Detailed Description

Updates the Keil μ Vision configuration, with the correct include paths and overlays.

The list of overlays is generated using the [overlays.awk](#) script and the configuration is updated using the [xml.awk](#) script.

Index

- any
 - depends.awk, 46
- awk2doxygen.awk
 - debug, 7
 - filter103, 9
 - filter115, 9
 - filter14, 7
 - filter178, 9
 - filter185, 9
 - filter196, 10
 - filter203, 10
 - filter219, 10
 - filter226, 10
 - filter241, 10
 - filter254, 11
 - filter38, 8
 - filter84, 8
 - filter94, 9
 - genFunction, 8
 - initDoc, 8
- buffer
 - dbc2c.awk, 28
- build.awk
 - compact, 15
 - esc, 15
 - filter118, 13
 - flags, 15
 - linklist, 17
 - obj_to_bin, 16
 - src_to_obj, 16
 - src_to_origin, 16
 - verbose, 14
 - verbose_keys, 14
 - verbose_values, 14
- cmdAttrib
 - xml.awk, 58
- cmdDelete
 - xml.awk, 59
- cmdDeleteAttrib
 - xml.awk, 59
- cmdInsert
 - xml.awk, 59
- cmdPrint
 - xml.awk, 59
- cmdRename
 - xml.awk, 58
- cmdRenameAttrib
 - xml.awk, 58
- cmdSearch
 - xml.awk, 57
- cmdSelect
 - xml.awk, 57
- cmdSelectInserted
 - xml.awk, 59
- cmdSet
 - xml.awk, 58
- compact
 - build.awk, 15
 - depends.awk, 47
- cstrip.awk
 - filter18, 18
 - filter44, 18
- dbc2c.awk
 - buffer, 28
 - debug, 28
 - error, 27
 - euclid, 37
 - fetch, 28
 - fetch_attrval, 35
 - fetchStr, 28
 - filter, 40
 - filter1434, 37
 - filter1841, 43
 - filter329, 27
 - filter439, 27
 - fsm_attr, 35
 - fsm_attrdefault, 34
 - fsm_attrrange, 34
 - fsm_comment, 33
 - fsm_discard, 30
 - fsm_ecu, 31
 - fsm_enum, 31
 - fsm_env, 32
 - fsm_env_data, 32
 - fsm_msg, 32
 - fsm_relattr, 35
 - fsm_relattrrange, 34
 - fsm_sig, 33
 - fsm_sig_enum, 31
 - fsm_siggrp, 36
 - fsm_start, 37
 - fsm_symbols, 36
 - fsm_tx, 36
 - getContext, 29
 - getUniqueEnum, 30
 - msgid, 43
 - msgidext, 43

- rational, 40
- rationalFmt, 38
- rationalD, 38
- rationalN, 38
- setTypes, 42
- siggrpident, 42
- sigident, 42
- strip, 29
- template, 41
- tpl_line, 41
- warn, 28
- whole, 29
- debug
 - awk2doxygen.awk, 7
 - dbc2c.awk, 28
- depends.awk
 - any, 46
 - compact, 47
 - extract, 46
 - filter184, 47
 - rescape, 45
 - sescape, 46
 - testf, 45
- empty
 - xml.awk, 56
- error
 - dbc2c.awk, 27
- esc
 - build.awk, 15
- escape
 - xml.awk, 56
- euclid
 - dbc2c.awk, 37
- explode
 - xml.awk, 56
- extract
 - depends.awk, 46
- fetch
 - dbc2c.awk, 28
- fetch_attrval
 - dbc2c.awk, 35
- fetchStr
 - dbc2c.awk, 28
- file2doxygen.awk
 - filter13, 48
 - filter22, 48
 - filter31, 48
- filter
 - dbc2c.awk, 40
- filter.sugar.awk
 - filter10, 49
 - filter15, 49
 - filter20, 50
 - filter26, 50
 - filter31, 50
- filter10
 - filter.sugar.awk, 49
- filter103
 - awk2doxygen.awk, 9
- filter104
 - overlays.awk, 53
- filter115
 - awk2doxygen.awk, 9
- filter117
 - overlays.awk, 53
- filter118
 - build.awk, 13
- filter13
 - file2doxygen.awk, 48
- filter130
 - overlays.awk, 53
- filter14
 - awk2doxygen.awk, 7
- filter1434
 - dbc2c.awk, 37
- filter15
 - filter.sugar.awk, 49
- filter152
 - overlays.awk, 53
- filter178
 - awk2doxygen.awk, 9
- filter18
 - cstrip.awk, 18
- filter184
 - depends.awk, 47
- filter1841
 - dbc2c.awk, 43
- filter185
 - awk2doxygen.awk, 9
- filter19
 - overlays.awk, 51
- filter196
 - awk2doxygen.awk, 10
- filter20
 - filter.sugar.awk, 50
- filter203
 - awk2doxygen.awk, 10
- filter219
 - awk2doxygen.awk, 10
- filter22
 - file2doxygen.awk, 48
- filter226
 - awk2doxygen.awk, 10
- filter241
 - awk2doxygen.awk, 10
- filter254
 - awk2doxygen.awk, 11
- filter26
 - filter.sugar.awk, 50
- filter31
 - file2doxygen.awk, 48
 - filter.sugar.awk, 50
- filter329
 - dbc2c.awk, 27
- filter34

- xml.awk, 56
- filter38
 - awk2doxygen.awk, 8
- filter439
 - dbc2c.awk, 27
- filter44
 - cstrip.awk, 18
- filter49
 - overlays.awk, 51
- filter56
 - overlays.awk, 52
- filter567
 - xml.awk, 60
- filter63
 - overlays.awk, 52
- filter631
 - xml.awk, 60
- filter70
 - overlays.awk, 52
- filter80
 - overlays.awk, 52
- filter84
 - awk2doxygen.awk, 8
- filter89
 - overlays.awk, 52
- filter94
 - awk2doxygen.awk, 9
- flags
 - build.awk, 15
- fsm_attr
 - dbc2c.awk, 35
- fsm_attrdefault
 - dbc2c.awk, 34
- fsm_attrrange
 - dbc2c.awk, 34
- fsm_comment
 - dbc2c.awk, 33
- fsm_discard
 - dbc2c.awk, 30
- fsm_ecu
 - dbc2c.awk, 31
- fsm_enum
 - dbc2c.awk, 31
- fsm_env
 - dbc2c.awk, 32
- fsm_env_data
 - dbc2c.awk, 32
- fsm_msg
 - dbc2c.awk, 32
- fsm_relattr
 - dbc2c.awk, 35
- fsm_relattrrange
 - dbc2c.awk, 34
- fsm_sig
 - dbc2c.awk, 33
- fsm_sig_enum
 - dbc2c.awk, 31
- fsm_siggrp
 - dbc2c.awk, 36
- fsm_start
 - dbc2c.awk, 37
- fsm_symbols
 - dbc2c.awk, 36
- fsm_tx
 - dbc2c.awk, 36
- genFunction
 - awk2doxygen.awk, 8
- getContext
 - dbc2c.awk, 29
- getUniqueEnum
 - dbc2c.awk, 30
- initDoc
 - awk2doxygen.awk, 8
- linklist
 - build.awk, 17
- Makefile, 5
- Makefile.include, 6
- msgid
 - dbc2c.awk, 43
- msgidext
 - dbc2c.awk, 43
- obj_to_bin
 - build.awk, 16
- overlays.awk
 - filter104, 53
 - filter117, 53
 - filter130, 53
 - filter152, 53
 - filter19, 51
 - filter49, 51
 - filter56, 52
 - filter63, 52
 - filter70, 52
 - filter80, 52
 - filter89, 52
- printNode
 - xml.awk, 60
- rational
 - dbc2c.awk, 40
- rationalFmt
 - dbc2c.awk, 38
- rationalD
 - dbc2c.awk, 38
- rationalN
 - dbc2c.awk, 38
- rescape
 - depends.awk, 45
- scripts/awk2doxygen.awk, 6
- scripts/build.awk, 11
- scripts/cstrip.awk, 17

- scripts/dbc.sh, 18
- scripts/dbc2c.awk, 19
- scripts/depends.awk, 44
- scripts/file2doxygen.awk, 48
- scripts/filter.sugar.awk, 49
- scripts/overlays.awk, 50
- scripts/sdcc.sh, 53
- scripts/testver.sh, 54
- scripts/xml.awk, 54
- sescape
 - depends.awk, 46
- setTypes
 - dbc2c.awk, 42
- siggrpident
 - dbc2c.awk, 42
- sigident
 - dbc2c.awk, 42
- src_to_obj
 - build.awk, 16
- src_to_origin
 - build.awk, 16
- strip
 - dbc2c.awk, 29

- template
 - dbc2c.awk, 41
- testf
 - depends.awk, 45
- tpl_line
 - dbc2c.awk, 41

- uVisionupdate.sh, 61

- verbose
 - build.awk, 14
- verbose_keys
 - build.awk, 14
- verbose_values
 - build.awk, 14

- warn
 - dbc2c.awk, 28
- whole
 - dbc2c.awk, 29

- xml.awk
 - cmdAttrib, 58
 - cmdDelete, 59
 - cmdDeleteAttrib, 59
 - cmdInsert, 59
 - cmdPrint, 59
 - cmdRename, 58
 - cmdRenameAttrib, 58
 - cmdSearch, 57
 - cmdSelect, 57
 - cmdSelectInserted, 59
 - cmdSet, 58
 - empty, 56
 - escape, 56
 - explode, 56
 - filter34, 56
 - filter567, 60
 - filter631, 60
 - printNode, 60