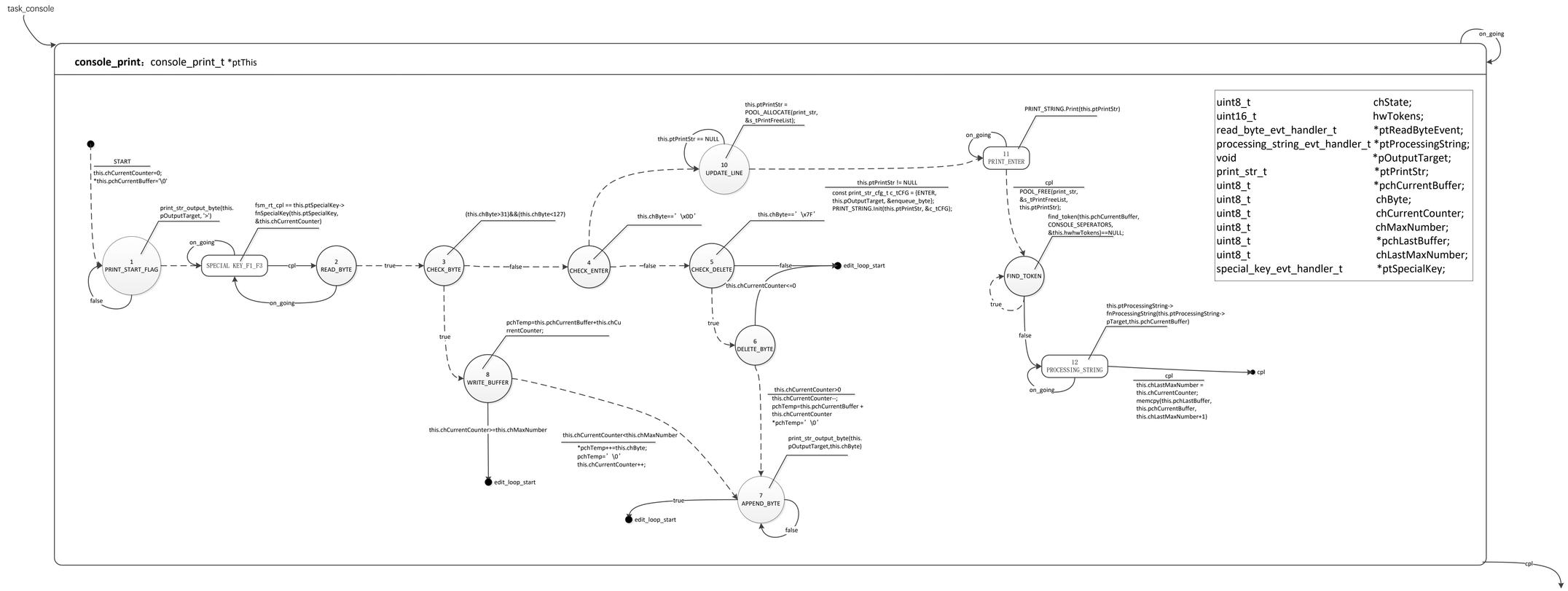
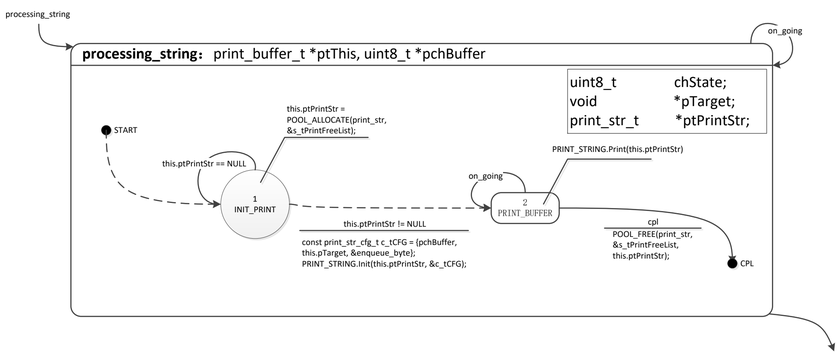
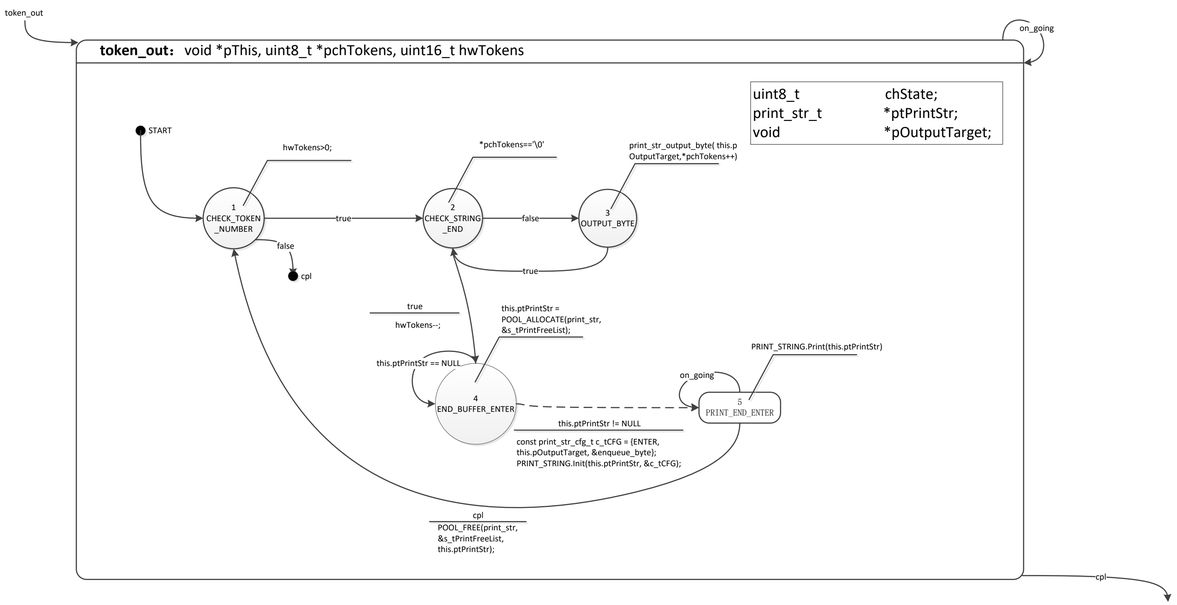

task_console
console_print：console_print_t *ptThis
on_going
START
this.chCurrentCounter=0;
*this.pchCurrentBuffer='\0'
1
PRINT_START_FLAG
false
print_str_output_byte(this.
pOutputTarget, '>')
on_going
SPECIAL KEY_F1_F3
fsm_rt_cpl == this.ptSpecialKey->
fnSpecialKey(this.ptSpecialKey,
&this.chCurrentCounter)
cpl
2
READ_BYTE
on_going
true
(this.chByte>31)&&(this.chByte<127)
3
CHECK_BYTE
false
this.chByte==' \x0D'
4
CHECK_ENTER
false
this.chByte==' \x7F'
5
CHECK_DELETE
false
edit_loop_start
this.chCurrentCounter<=0
true
6
DELETE_BYTE
true
pchTemp=this.pchCurrentBuffer+this.chCu
rrentCounter;
8
WRITE_BUFFER
this.chCurrentCounter>=this.chMaxNumber
edit_loop_start
this.chCurrentCounter<this.chMaxNumber
*pchTemp++=this.chByte;
pchTemp=' \0'
this.chCurrentCounter++;
this.chCurrentCounter>0
this.chCurrentCounter--;
pchTemp=this.pchCurrentBuffer +
this.chCurrentCounter
*pchTemp=' \0'
print_str_output_byte(this.
pOutputTarget,this.chByte)
7
APPEND_BYTE
true
edit_loop_start
false
this.ptPrintStr =
POOL_ALLOCATE(print_str,
&s_tPrintFreeList);
this.ptPrintStr == NULL
10
UPDATE_LINE
this.ptPrintStr != NULL
const print_str_cfg_t c_tCFG = {ENTER,
this.pOutputTarget, &enqueue_byte};
PRINT_STRING.Init(this.ptPrintStr, &c_tCFG);
PRINT_STRING.Print(this.ptPrintStr)
on_going
11
PRINT_ENTER
cpl
POOL_FREE(print_str,
&s_tPrintFreeList,
this.ptPrintStr);
find_token(this.pchCurrentBuffer,
CONSOLE_SEPERATORS,
&this.hwhwTokens)==NULL;
FIND_TOKEN
true
false
this.ptProcessingString->
fnProcessingString(this.ptProcessingString->
pTarget,this.pchCurrentBuffer)
12
PROCESSING_STRING
on_going
cpl
cpl
this.chLastMaxNumber =
this.chCurrentCounter;
memcpy(this.pchLastBuffer,
this.pchCurrentBuffer,
this.chLastMaxNumber+1)
cpl
uint8_t chState;
uint16_t hwTokens;
read_byte_evt_handler_t *ptReadByteEvent;
processing_string_evt_handler_t *ptProcessingString;
void *pOutputTarget;
print_str_t *ptPrintStr;
uint8_t *pchCurrentBuffer;
uint8_t chByte;
uint8_t chCurrentCounter;
uint8_t chMaxNumber;
uint8_t *pchLastBuffer;
uint8_t chLastMaxNumber;
special_key_evt_handler_t *ptSpecialKey;


processing_string
processing_string：print_buffer_t *ptThis, uint8_t *pchBuffer
on_going
uint8_t chState;
void *pTarget;
print_str_t *ptPrintStr;
START
this.ptPrintStr =
POOL_ALLOCATE(print_str,
&s_tPrintFreeList);
this.ptPrintStr == NULL
1
INIT_PRINT
this.ptPrintStr != NULL
const print_str_cfg_t c_tCFG = {pchBuffer,
this.pTarget, &enqueue_byte};
PRINT_STRING.Init(this.ptPrintStr, &c_tCFG);
on_going
2
PRINT_BUFFER
PRINT_STRING.Print(this.ptPrintStr)
cpl
POOL_FREE(print_str,
&s_tPrintFreeList,
this.ptPrintStr);
CPL


token_out
on_going
token_out：void *pThis, uint8_t *pchTokens, uint16_t hwTokens
uint8_t chState;
print_str_t *ptPrintStr;
void *pOutputTarget;
START
hwTokens>0;
1
CHECK_TOKEN
_NUMBER
true
false
cpl
*pchTokens=='\0'
2
CHECK_STRING
_END
false
print_str_output_byte(this.p
OutputTarget,*pchTokens++)
3
OUTPUT_BYTE
true
true
hwTokens--;
this.ptPrintStr =
POOL_ALLOCATE(print_str,
&s_tPrintFreeList);
this.ptPrintStr == NULL
4
END_BUFFER_ENTER
this.ptPrintStr != NULL
const print_str_cfg_t c_tCFG = {ENTER,
this.pOutputTarget, &enqueue_byte};
PRINT_STRING.Init(this.ptPrintStr, &c_tCFG);
on_going
PRINT_STRING.Print(this.ptPrintStr)
5
PRINT_END_ENTER
cpl
POOL_FREE(print_str,
&s_tPrintFreeList,
this.ptPrintStr);
cpl