

master ▾

...

usbx / common / usbx_host_classes / src / ux_host_class_cdc_ecm_mac_address_get.c



TiejunMS Release 6.2.0 ✖

History

3 contributors



355 lines (279 sloc) | 19.8 KB

...

```

1  /*****
2  /*
3  /*      Copyright (c) Microsoft Corporation. All rights reserved.      */
4  /*
5  /*      This software is licensed under the Microsoft Software License  */
6  /*      Terms for Microsoft Azure RTOS. Full text of the license can be  */
7  /*      found in the LICENSE file at https://aka.ms/AzureRTOS_EULA      */
8  /*      and in the root directory of this software.                      */
9  /*
10 /*****
11
12
13 /*****
14 /*****
15 /**                                  */
16 /** USBX Component                  */
17 /**                                  */
18 /** CDC ECM Class                   */
19 /**                                  */
20 /*****
21 /*****
22
23
24 /* Include necessary system files. */
25
26 #define UX_SOURCE_CODE
27
28 #include "ux_api.h"
29 #include "ux_host_class_cdc_ecm.h"

```

```

30 #include "ux_host_stack.h"
31
32
33 /*****
34  */
35  /* FUNCTION                                RELEASE */
36  /*
37  /*  _ux_host_class_cdc_ecm_mac_address_get          PORTABLE C */
38  /*
39  /* 6.2.0 */
40  /* AUTHOR */
41  /*
42  /* Chaoqiong Xiao, Microsoft Corporation */
43  /*
44  /* DESCRIPTION */
45  /* This function calls the USBX stack to retrieve the MAC address from */
46  /* the configuration descriptor. */
47  /*
48  /* INPUT */
49  /*
50  /* cdc_ecm Pointer to cdc_ecm class */
51  /*
52  /* OUTPUT */
53  /*
54  /* Completion Status */
55  /*
56  /* CALLS */
57  /*
58  /* _ux_host_stack_transfer_request Transfer request */
59  /* _ux_utility_memory_allocate Allocate memory */
60  /* _ux_utility_memory_free Free memory */
61  /* _ux_utility_descriptor_parse Parse descriptors */
62  /*
63  /* CALLED BY */
64  /*
65  /* _ux_host_class_cdc_ecm_activate CDC ECM class activate */
66  /*
67  /* RELEASE HISTORY */
68  /*
69  /* DATE NAME DESCRIPTION */
70  /*
71  /* 05-19-2020 Chaoqiong Xiao Initial Version 6.0 */
72  /* 09-30-2020 Chaoqiong Xiao Modified comment(s), */
73  /* resulting in version 6.1 */
74  /* 07-29-2022 Chaoqiong Xiao Modified comment(s), */
75  /* checked MAC string length, */
76  /* resulting in version 6.1.12 */
77  /* 10-31-2022 Chaoqiong Xiao Modified comment(s), */
78  /* checked descriptor length, */

```

```

79      /*                      resulting in version 6.2.0 */
80      /*                      */
81      /*****
82      UINT  _ux_host_class_cdc_ecm_mac_address_get(UX_HOST_CLASS_CDC_ECM *cdc_ecm)
83      {
84
85          UINT                      status;
86          UX_ENDPOINT              *control_endpoint;
87          UX_TRANSFER              *transfer_request;
88          UX_CONFIGURATION_DESCRIPTOR configuration_descriptor;
89          UCHAR                    *descriptor;
90          UCHAR                    *start_descriptor = UX_NULL;
91          ULONG                    configuration_index;
92          ULONG                    total_configuration_length;
93          UINT                     descriptor_length;
94          UINT                     descriptor_type;
95          UINT                     descriptor_subtype;
96          UX_HOST_CLASS_ECM_INTERFACE_DESCRIPTOR ecm_interface_descriptor;
97          UCHAR                    *mac_address_string;
98          ULONG                    string_index;
99          ULONG                    string_length;
100         UCHAR                    element_content;
101         UCHAR                    element_hexa_upper;
102         UCHAR                    element_hexa_lower;
103
104         /* We now need to retrieve the MAC address of the node which is embedded in the ECM descriptor
105            We will parse the entire configuration descriptor of the device and look for the ECM Ethern
106            configuration_index = cdc_ecm -> ux_host_class_cdc_ecm_interface_data -> ux_interface_configur
107
108         /* We need to get the default control endpoint transfer request pointer. */
109         control_endpoint = &cdc_ecm -> ux_host_class_cdc_ecm_device -> ux_device_control_endpoint;
110         transfer_request = &control_endpoint -> ux_endpoint_transfer_request;
111
112         /* Need to allocate memory for the descriptor. Since we do not know the size of the
113            descriptor, we first read the first bytes. */
114         descriptor = _ux_utility_memory_allocate(UX_SAFE_ALIGN, UX_CACHE_SAFE_MEMORY, UX_CONFIGURATIO
115         if (descriptor == UX_NULL)
116             return(UX_MEMORY_INSUFFICIENT);
117
118         /* Memorize the descriptor start address. */
119         start_descriptor = descriptor;
120
121         /* Create a transfer request for the GET_DESCRIPTOR request. */
122         transfer_request -> ux_transfer_request_data_pointer = descriptor;
123         transfer_request -> ux_transfer_request_requested_length = UX_CONFIGURATION_DESCRIPTOR_LENGTH
124         transfer_request -> ux_transfer_request_function = UX_GET_DESCRIPTOR;
125         transfer_request -> ux_transfer_request_type = UX_REQUEST_IN | UX_REQUEST_TYPE_ST
126         transfer_request -> ux_transfer_request_value = (UX_CONFIGURATION_DESCRIPTOR_ITEM
127         transfer_request -> ux_transfer_request_index = 0;

```

```

128
129     /* Send request to HCD layer. */
130     status = _ux_host_stack_transfer_request(transfer_request);
131
132     /* Check for correct transfer and entire descriptor returned. */
133     if ((status == UX_SUCCESS) && (transfer_request -> ux_transfer_request_actual_length == UX_CON
134     {
135
136         /* Parse the descriptor so that we can read the total length. */
137         _ux_utility_descriptor_parse(descriptor, _ux_system_configuration_descriptor_structure,
138                                         UX_CONFIGURATION_DESCRIPTOR_ENTRIE
139
140         /* We don't need this descriptor now. */
141         _ux_utility_memory_free(descriptor);
142
143         /* Reallocate the memory necessary for the reading the entire descriptor. */
144         total_configuration_length = configuration_descriptor.wTotalLength;
145         descriptor = _ux_utility_memory_allocate(UX_SAFE_ALIGN, UX_CACHE_SAFE_MEMORY, total_conf
146         if (descriptor == UX_NULL)
147             return(UX_MEMORY_INSUFFICIENT);
148
149         /* Save this descriptor address. */
150         start_descriptor = descriptor;
151
152         /* Read the descriptor again with the correct length this time. */
153         transfer_request -> ux_transfer_request_requested_length = total_configuration_length;
154
155         /* Since the address of the descriptor may have changed, reprogram it. */
156         transfer_request -> ux_transfer_request_data_pointer = descriptor;
157
158         /* Send request to HCD layer. */
159         status = _ux_host_stack_transfer_request(transfer_request);
160
161         /* Check for correct transfer and entire descriptor returned. */
162         if ((status == UX_SUCCESS) && (transfer_request -> ux_transfer_request_actual_length == co
163         {
164
165             /* The ECM descriptor is embedded within the configuration descriptor. We parse the
166             entire descriptor to locate the ECM functional descriptor portion. */
167             while (total_configuration_length)
168             {
169
170                 /* Gather the length and type of the descriptor. */
171                 descriptor_length = *descriptor;
172                 descriptor_type = *(descriptor + 1);
173                 descriptor_subtype = *(descriptor + 2);
174
175                 /* Descriptor length validation. */
176                 if (descriptor_length < 3 || descriptor_length > total_configuration_length)

```

```

177     {
178
179         /* Error trap. */
180         _ux_system_error_handler(UX_SYSTEM_LEVEL_THREAD, UX_SYSTEM_CONTEXT_CLASS, UX_D
181
182         /* Free descriptor memory. */
183         _ux_utility_memory_free(start_descriptor);
184
185         /* Return error. */
186         return(UX_DESCRIPTOR_CORRUPTED);
187     }
188
189     /* Check the type for an interface descriptor and the subtype for a ECM functional
190     if ((descriptor_type == UX_HOST_CLASS_CDC_ECM_CS_INTERFACE) && (descriptor_subtype
191     {
192
193         /* Parse the interface descriptor and make it machine independent. */
194         _ux_utility_descriptor_parse(descriptor,
195             _ux_system_ecm_interface_descriptor_structure,
196             UX_HOST_CLASS_CDC_ECM_INTERFACE_DESCRIPTOR_ENTRIES,
197             (UCHAR *) &ecm_interface_descriptor);
198
199
200         /* Release the memory. */
201         _ux_utility_memory_free(start_descriptor);
202
203         /* We now have the ECM functional descriptor in memory. We can retrieve the in
204         which we need for NetX. */
205
206         /* Allocate memory for the MAC address. */
207         mac_address_string = _ux_utility_memory_allocate(UX_SAFE_ALIGN, UX_CACHE_SAFE
208
209         /* Check memory allocation. */
210         if (mac_address_string == UX_NULL)
211             return(UX_MEMORY_INSUFFICIENT);
212
213         /* Create a transfer request for the GET_DESCRIPTOR request. */
214         transfer_request -> ux_transfer_request_data_pointer = mac_address_string
215         transfer_request -> ux_transfer_request_requested_length = UX_HOST_CLASS_CDC_
216         transfer_request -> ux_transfer_request_function = UX_GET_DESCRIPTOR;
217         transfer_request -> ux_transfer_request_type = UX_REQUEST_IN | UX
218         transfer_request -> ux_transfer_request_value = (UX_STRING_DESCRIP
219         transfer_request -> ux_transfer_request_index = 0x0409;
220
221         /* Send request to HCD layer. */
222         status = _ux_host_stack_transfer_request(transfer_request);
223
224         /* Check for correct transfer. */
225         if (status == UX_SUCCESS)

```

```

226     {
227
228         /* Translate from Unicode to string. Length is in the first byte followed
229         We must take away 2 from it and divide by 2 to find the right ascii len
230         string_length = (ULONG) *mac_address_string;
231
232         /* Check the length of the MAC address Unicode string
233         (length or 1B + type of 1B + string or 12*2B). */
234         if (string_length != 26)
235         {
236
237             /* Error trap. */
238             _ux_system_error_handler(UX_SYSTEM_LEVEL_THREAD, UX_SYSTEM_CONTEXT_CLA
239
240             /* Return error. */
241             status = UX_DESCRIPTOR_CORRUPTED;
242         }
243         else
244         {
245
246             /* No error in length, decode the string. */
247             string_length -= 2;
248             string_length = string_length / 2;
249
250             /* Now we have a string of 12 hex ASCII digits to be translated into 6
251             and copy into the node ID. */
252             for (string_index = 0; string_index < string_length; string_index++)
253             {
254
255                 /* Get the upper element from the ASCII string. */
256                 element_content = *(mac_address_string + (string_index * 2) + 2);
257
258                 /* We have a valid element content. Turn it into a hex decimal va
259                 that only hex digits are allowed. */
260                 if (element_content <= '9')
261
262                     /* We have a digit. */
263                     element_hexa_upper = (UCHAR)(element_content - '0');
264
265                 else
266                 {
267                     /* We have a 'A' to 'F' or 'a' to 'f' value. */
268                     if (element_content >= 'a')
269
270                         /* We have a 'a' to 'f' char. */
271                         element_hexa_upper = (UCHAR)(element_content - 'a' + 10);
272
273                     else
274

```

```

275         /* We have a 'A' to 'F' char. */
276         element_hexa_upper = (UCHAR)(element_content - 'A' + 10);
277
278     }
279
280     /* Get the lower element from the ASCII string. */
281     element_content = *(mac_address_string + ((string_index + 1) * 2)
282
283     /* We have a valid element content. Turn it into a hexa decimal v
284     that only hex digits are allowed. */
285     if (element_content <= '9')
286
287         /* We have a digit. */
288         element_hexa_lower = (UCHAR)(element_content - '0');
289
290     else
291     {
292         /* We have a 'A' to 'F' or 'a' to 'f' value. */
293         if (element_content >= 'a')
294
295             /* We have a 'a' to 'f' char. */
296             element_hexa_lower = (UCHAR)(element_content - 'a' + 10);
297
298         else
299
300             /* We have a 'A' to 'F' char. */
301             element_hexa_lower = (UCHAR)(element_content - 'A' + 10);
302
303     }
304
305     /* Assemble the byte from the 2 nibbles and store it into the node
306     *(cdc_ecm -> ux_host_class_cdc_ecm_node_id + string_index / 2) = (
307
308     /* Skip the lower nibble. */
309     string_index ++;
310
311 }
312
313 /* Operation was successful ! */
314 status = UX_SUCCESS;
315 }
316 }
317 else
318 {
319
320     /* We have a bad MAC address string. Do not proceed. */
321     status = UX_ERROR;
322 }
323

```

```
324         /* Free the MAC address string. */
325         _ux_utility_memory_free(mac_address_string);
326
327         /* Return completion status. */
328         return(status);
329     }
330     else
331     {
332
333         /* Jump to the next descriptor if we have not reached the end. */
334         descriptor += descriptor_length;
335
336         /* And adjust the length left to parse in the descriptor. */
337         total_configuration_length -= descriptor_length;
338     }
339 }
340 }
341 }
342
343 /* Error trap. */
344 _ux_system_error_handler(UX_SYSTEM_LEVEL_THREAD, UX_SYSTEM_CONTEXT_CLASS, UX_DESCRIPTOR_CORRUPTED);
345
346 /* If trace is enabled, insert this event into the trace buffer. */
347 UX_TRACE_IN_LINE_INSERT(UX_TRACE_ERROR, UX_DESCRIPTOR_CORRUPTED, &configuration_descriptor, 0,
348
349 /* Release the memory. */
350 _ux_utility_memory_free(start_descriptor);
351
352 /* Return an error. */
353 return(UX_DESCRIPTOR_CORRUPTED);
354
355 }
```