CANdesc

Technical Reference

GM / Opel specifics

Version 3.1.0

Authors	Mishel Shishmanyan; Christoph Rätz; Oliver Garnatz; Matthias Heil; Katrin Thurow
Status	**Released**

1 History

Author	Date	Version	Remarks
Mishel Shishmanyan	2002-05-07	0.9.0	Creation
Christoph Rätz	2002-07-31	0.9.4	Reworked and released
Mishel Shishmanyan	2002-12-12	1.0.0	Added requirements for ProgrammingMode (Sid $A5) and DeviceControl(Sid $AE) Released
Mishel Shishmanyan	2003-04-09	1.1.0	Added default CANdelaStudio attribute settings for GM/OPEL ECUs
Oliver Garnatz	2003-12-19	2.0.0	Adapted to CANdesc 2.xx.xx New Word template used.
Oliver Garnatz	2004-05-14	2.1.0	Replaced AppDesc with ApplDesc Changed support level of ‘Security access’
Oliver Garnatz, Mishel Shishmanyan	2004-07-15	2.2.0	Added description of CANdesc OBD support
Mishel Shishmanyan	2006-05-02	2.3.0	Added: - Service DynamicallyDefineMessage ($2C) - Service DefinePIDByAddress ($2D) - The PacketHandler (another type of service processor) Modified: - Cosmetics - All APIs described in detailed table form Removed: - None
Jason Wolbers	2006-08-03	2.4.0	Improved wording
Mishel Shishmanyan	2006-10-22	2.5.0	Added: - 6.1 *“ECU Address configuration”*
Mishel Shishmanyan	2006-11-03	2.6.0	Modified: - 6.1.2 *“Multi address ECU (dynamic addressing)”*
Mishel Shishmanyan	2007-12-14	2.7.0	Modified: - 8.3 *“Service attributes”*

			- *7.6 "Service DisableNormalCommunication ($28)"* Removed: - 6.1.2 "Multi address ECU (dynamic addressing)" – only target addresses 0xFE and 0xFD (for gateways only) are accepted by CANdesc.
Mishel Shishmanyan	2010-12-21	2.8.0	Modified: - *6.1 ECU Address configuration* Added: - *7.5 Service SecurityAccess ($27)*
Matthias Heil	2011-04-20	3.0.0	Modified: - Update to new formatting - *7.11 Service ProgrammingMode ($A5)* Added: - *5.3 Update from earlier versions* - *8.4 State group for the Programming Sequence*
Katrin Thurow	2011-12-16	3.1.0	Modified: *6.6.3 High speed programming mode state*

Contents

2 Related documents

- Technical Reference CANdesc
- User Manual CANdesc

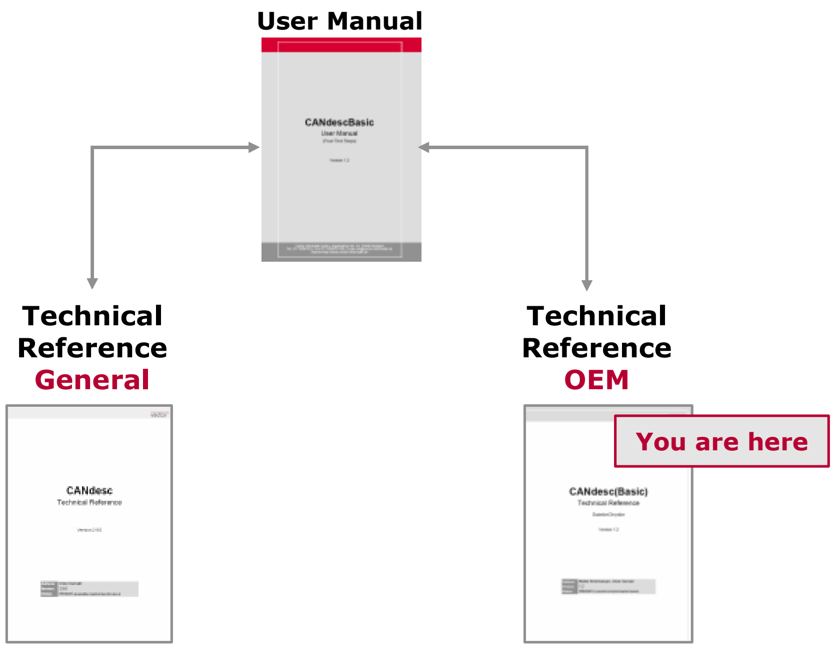


Figure 2-1 Manuals and References for CANdesc

All GM/Opel specific CANdesc topics are described within this technical reference. Topics which are common to all OEMs (e.g. features, concepts) are located in the general technical reference document “TechnicalReference_CANdesc”.

For faster integration, please refer to the user manual document “UserManual_CANdesc”.

3 Overview

The GM/Opel version of CANdesc uses an internal implementation to handle many diagnostic tasks. Many of these tasks are realized through generated code (constants, state count, etc.) which gives more flexibility to the application in case of specification or variant changes. On the other hand, there are “built-in” diagnostic service implementations which can free the application from some very complex tasks (e.g. ReadDiagnosticInformation (SID $A9), ReadDataByPacketIdentifier (SID $AA), etc.) and hide all of the underlying functionality under a simple signal interface for the application. In addition, CANdesc also handles certain GM/Opel specific management functionality (e.g. virtual networks) in order to fully comply with GM/Opel diagnostic specifications.

4 CANdesc support by diagnostic service

CANdesc provides three possible levels of support independently for each diagnostic service – complete, assisted, or basic. The level of support varies according to CANdesc capability and user selection in CANdelaStudio. All three levels of support provide complete communication handling (including all transport protocol processing and error handling), diagnostic session and timer management, and basic error checking.

Communication handling not only includes testing support of service, but also consistency of service, sub-function and/or identifier combination. A validity check of the addressing method and data length is performed. Parallel requests are managed (SID $3E, $AA, and $A9) when allowed. As error handling is a significant part of any ECU software, all low level errors are handled internally by CANdesc. Finally, CANdesc assists with the connection to GMLAN (e.g. supervision and management of the VN timer, message transmission mode, bus speed switching, etc.).

Complete

Complete support means that CANdesc is capable of handling the diagnostic transaction without requesting support from the ECU application. The ECU developer need not provide any code to help implement the diagnostic feature; CANdesc handles all processing. In the case where diagnostic messages contain real-time data, or “signals”, CANdesc can map that data to global variables in the ECU application and read/write the values directly to/from RAM without calling any ECU application callbacks; the ECU developer does not have to concern himself with the protocol level implementation. If a service modifies a diagnostic state group, CANdesc notifies the application using a callback function.

Assisted

Assisted support means that CANdesc is capable of fully parsing request messages and building response messages, but it does not contain the logic necessary to execute the request or determine data values. The ECU developer must provide callbacks for CANdesc to fill these logic gaps, which typically come in the form of calculating a signal value, controlling an I/O port, or perhaps executing an ECU-specific feature such as a self-test or EEPROM access routine.

Basic

Basic support means that CANdesc is only capable of identifying that the ECU application must process the request. The ECU application may have to provide logic to validate the request message and build the response byte-by-byte. This level of support is only used where code generation is not practical or supported.

RequestCurrentDiagnosticData ($01) – Assisted

The application must implement only the handling of this service (no request length validation).

RequestFreezeFrameData ($02) – Assisted

The application must implement only the handling of this service (no request length validation).

RequestEmissionRelatedDTC ($03) – Assisted

The application must implement only the handling of this service (no request length validation).

ClearDiagnosticInformation ($04) – Assisted

The application must provide a function that clears fault memory.

RequestTestResultsForNonContinouslyMonitoredSystems ($06) – Assisted

The application must implement only the handling of this service (no request length validation).

RequestTestResultsForContinouslyMonitoredSystems ($07) – Assisted

The application must implement only the handling of this service (no request length validation).

RequestControlOfOnBoardSystemTestOrComponent ($08) – Assisted

The application must implement only the handling of this service (no request length validation when the request data length is a constant value).

RequestVehicleInformation ($09) – Assisted

The application must implement only the handling of this service (no request length validation).

InitiateDiagnosticOperation ($10) – Assisted

The application must provide a function that implements the logic to enable/disable the settings of DTCs.

ReadFailureRecordData ($12) – Basic

The application must provide a function that implements the request parsing (validation) and the logic to report the failure record data.

ReadDataByIdentifier ($1A) – Complete

CANdesc completely implements this service for DIDs whose data maps to global variables. Assisted support is provided for DIDs whose data does not map to global variables.

ReturnToNormalMode ($20) – Complete

The application must provide an implementation for the event-based callbacks triggered by this service.

ReadDataByParameterIdentifier ($22) – Complete

CANdesc completely implements this service for PIDs whose data maps to global variables. Assisted support is provided for PIDs whose data does not map to global variables. In any case, multiple PID handling (in a single request) is handled internally by CANdesc.

ReadMemoryByAddress ($23) – Basic

The application must provide a function to determine if the requested address is valid and a function to return the data stored at the requested address.

SecurityAccess ($27) – Basic

The application must provide functions, which implement the security access mechanism.

By utilizing a diagnostic state group, CANdesc can track the current security level and assist the application in determining if a request is allowed at the given time.

DisableNormalCommunication ($28) – Complete

CANdesc completely implements this service.

DynamicallyDefineMessage ($2C) – Complete

CANdesc completely implements this service.

DefinePIDByAddress ($2D) – Complete

CANdesc completely implements this service.

RequestDownload ($34) – Basic

The application must implement this service.

TransferData ($36) – Basic

The application must implement this service.

WriteDataByIdentifier ($3B) – Complete

CANdesc completely implements this service for DIDs whose data maps to global variables. Assisted support is provided for DIDs whose data does not map to global variables.

TesterPresent ($3E) – Complete

CANdesc completely implements this service.

ReportProgrammedState ($A2) – Complete

CANdesc completely implements this service if the programmed state maps to a global variable. Assisted support is provided if the programmed state does not map to a global variable.

ProgrammingMode ($A5) – Assisted

The application must only provide functions that implement programming mode requests. CANdesc performs state checking internally.

ReadDiagnosticInformation ($A9) – Assisted

The application must provide functions that read fault memory and construct the response messages byte-by-byte. The application must provide functions to retrieve the number of fault codes and to step through the list of fault codes matching the requested mask. Moreover, it has to detect a change in the number of DTCs. CANdesc handles the scheduler internally.

ReadDataByPacketIdentifier ($AA) – Complete

CANdesc completely implements this service for DPIDs whose data maps to global variables. Assisted support is provided for DPIDs whose data does not map to global variables. The scheduler is handled internally by CANdesc.

DeviceControl ($AE) – Assisted

The application must provide device-specific functions that implement the control algorithms.

Caution
Diagnostic services other than those listed above are not supported by CANdesc in any way and must be implemented entirely by the ECU developer as a workaround.

5 Important application requirements

5.1 Initialization

In order to initialize the GM/Opel CANdesc component, the application must call the following function:

Prototype	
`void DescInitPowerOn (DescInitParam initParameter)`	
Parameter	
`initParameter`	Initialization parameter (recommended: 'kDescPowerOnInitParam')
Return code	
–	-
Functional Description	
Power-on initialization of the CANdesc component. The application must call this function once after power-on, before all other CANdesc functions. The GM/Opel version of CANdesc has no special behavior for initialization; therefore, the initialization function can be called with any parameter value. Even so, it is recommended that the ECU developer use 'kDescPowerOnInitParam' for the parameter value.	
Particularities and Limitations	
▶ **DescInitPowerOn (initParameter)** must be called after **TpInitPowerOn()** (please refer to the transport protocol documentation) or any reserved diagnostic connection will be lost. ▶ **DescInitPowerOn (initParameter)** calls **DescInit()** internally for further initializations	
Call context	
▶ Background-loop level with global interrupts disabled	

Table 5-1 DescInitPowerOn

Prototype	
`void DescInit (DescInitParam initParameter)`	
Parameter	
`initParameter`	Initialization parameter (recommended: 'kDescPowerOnInitParam')
Return code	
–	-
Functional Description	
Re-initializes the CANdesc component. The application can call this function to re-initialize CANdesc (e.g. after wakeup). All internal states will be set to their default values. The GM/Opel version of CANdesc has no special behavior for initialization; therefore, the initialization function can be called with any parameter value. Even so, it is recommended that the ECU developer use 'kDescPowerOnInitParam' for the parameter value.	
Particularities and Limitations	
▶ The application has already initialized CANdesc once by calling **DescInitPowerOn()**	
Call context	
▶ Background-loop level with global interrupts disabled	

Table 5-2 DescInit

5.2 DeviceControl ($AE) service requirement

The GM/Opel diagnostic specification requires that device control activity shall be limited by tester present timeout.

Please refer to the section *Service DeviceControl ($AE)* for more details.

5.3 Update from earlier versions

The behavior of the CANdesc embedded module has not changed fundamentally in CANdesc version 6.x, but the configuration is much more independent from the CDD settings. Many options that were previously only configurable as attributes in the CDD file are now available for configuration in the GENy configuration tool.

As part of this change, the naming scheme for service callbacks is now more independent from the CDD contents and more adherent to the GMW3110 specification. This will require modification of existing application code to fit the new naming scheme.

In addition, the implementation of mode $A5 was changed to use the standard CANdesc state management feature. You can now easily have service execution depend on the reprogramming sequence, e.g. prevent a service from executing while programming mode is active. Please also refer to chapter *7.11 - Service ProgrammingMode ($A5)* for more information.

6 GM/Opel specific functionality

6.1 ECU Address configuration

GM/Opel use extended addressing for the functional request message. By default, CANdesc will accept any request with target address 0xFE. There are some other use cases that are considered to be supported with the help of user configuration files (refer the "*TechnicalReference_CANdesc.pdf*" for configuration details).

6.1.1 Gateway ECUs

According to the GMW3110 v1.6, gateways shall be accessible also via the special target address 0xFD. To enable the reception on this address, please insert into your user configuration file for CANdesc the following definition:

```
#define DESC_ENABLE_GW_ECU_ADDR
```

6.1.2 Virtual network management

A GMLAN/IVLAN specific implementation is built into CANdesc, which completely integrates CANdesc into a GM/Opel project. CANdesc manages all aspects of the diagnostics VN, including activation of the VN in the GMLAN handler and then deactivation of the VN after diagnostic inactivity (e.g. missing tester) as described in GMW-3110.

When the first diagnostic request is received, CANdesc will activate the diagnostics VN to provide communication capability with the tester. The VN is then automatically deactivated under the following conditions:

- 8 seconds after the diagnostic request "ReturnToNormalMode" ($20) is received
- 8 seconds after the tester present timeout
- 8 seconds after the last response is sent, or in the case of requests that do not send a response, 8 seconds after the request is completely processed

6.1.3 Diagnostic activity notification

CANdesc notifies the application via a callback function when a diagnostic session begins (the first diagnostic request) and when it ends (the deactivation conditions listed above). These callbacks are listed below.

Prototype	
`void` **`ApplDescOnDiagActive`** `(void)`	
Parameter	
–	-
Return code	
–	-
Functional Description	
When the first diagnostic request (after ECU power-on or wakeup) is received, regardless of the addressing method, CANdesc calls this function to notify the application that ECU diagnostics are now active (so that the application can begin diagnostic specific tasks, for instance).	
Particularities and Limitations	
▶ None	
Call context	
▶ Interrupt context, if the CAN-driver is configured to handle receive messages in interrupt mode ▶ Background-loop level task context, if the CAN-driver is configured to handle receive messages in polling mode	

Table 6-1 ApplDescOnDiagActive

Prototype	
`void` **`ApplDescOnDiagInactive`** `(void)`	
Parameter	
–	-
Return code	
–	-
Functional Description	
Once the conditions for deactivating the diagnostics VN are met (i.e. the diagnostics VN timer reaches zero), CANdesc calls this function to notify the application that ECU diagnostics are no longer active (so that the application can end diagnostic specific tasks to lower CPU utilization, for instance).	
Particularities and Limitations	
▶ None	
Call context	
▶ Background-loop level task context of **DescTimerTask()**.	

Table 6-2 ApplDescOnDiagInactive

6.2 Request validation

The GM/Opel version of CANdesc is capable of performing the following request validations:

- Is the designed diagnostic buffer big enough to hold the whole request? If the request is physically addressed, the GM/Opel CANdesc implementation will accept it even if the length is greater than the defined buffer and let the “request length validation” routine handle the situation. If the request is functionally addressed and the length is greater than the defined buffer, it will be ignored (regardless of whether it would normally send a response).
- Is the requested SID supported? If the SID is not relevant for the ECU, CANdesc will automatically send a negative response with error code “ServiceNotSupported” ($11). Further processing will be aborted.
- Is the request addressing method for the SID correct? Two cases are possible:
  - The requested SID normally provides a response (as defined in CANdelaStudio). In this case, CANdesc will send a negative response with error code “ConditionsNotCorrect” ($22). Further processing will be aborted.
  - The requested SID normally does not provide a response (as defined in CANdelaStudio). In this case, the request will be ignored.
- Does the request meet the minimum required length (to be distinguishable from other instances)? Each request is formatted as shown in the figure below:

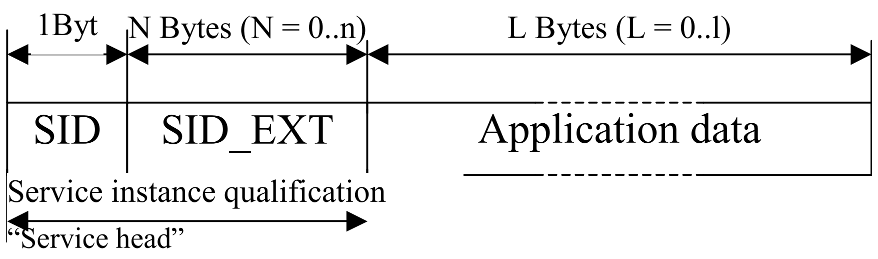


Figure 6-1 Request message logical format

  - In order to process the request further, the service instance must be found (which provides more detailed information about the request than the SID alone); therefore, the request must be at least n + 1 bytes in length, where n is service dependent.
  - If the length is less than the minimum allowed (as defined in CANdelaStudio), CANdesc will send a negative response with error code “SubfunctionNotSupportedInvalidFormat” ($12). Further processing will be aborted.
- Is the requested service instance supported by the ECU? If not, CANdesc will send a negative response with error code “SubfunctionNotSupportedInvalidFormat” ($12). Further processing will be aborted.
- Is the total request length correct? Two cases are possible:
  - The request length is dynamic. In this case, no automated check can be performed, and the task will be left to the application.

- The request length is fixed. In this case, CANdesc will check if the current request length matches the expected length (as defined in CANdelaStudio). If not, a negative response with error code “SubfunctionNotSupportedInvalidFormat” ($12) will be sent. Further processing will be aborted.

- Does the requested service instance have a defined pre-handler function (please refer to the user manual CANdesc document “UserManual_CANdesc” for more details about pre-handlers)? If so, it will be called. This allows the application to extend the built-in validation with additional custom checks.

Note
If the request passes all of the above validation checks, the main-handler is called (please refer to the user manual CANdesc for more details about main-handlers) for further processing.

6.3 Timeout events

6.3.1 Tester present timeout

Once the tester present timer has been activated, the tester must send the diagnostic service “Tester Present” ($3E) periodically in order to keep the timer running.

In case of a timeout, the diagnostic state will be initialized exactly the same as *Service ReturnToNormalMode ($20)*.

Note
CANdesc will automatically send an unsolicited positive response for the “ReturnToNormalMode” ($20) diagnostic service (see section *6.5 Sending an unsolicited single frame response*).

6.4 Using the extended negative response

The GM/Opel version of CANdesc supports the extended negative response format to specify service faults more precisely. This response has the following format:

$7F $AE $E3 $xx $yy (DeviceControlLimitsExceeded)

where $xx and $yy are the extended failure codes (as defined in an ECU specific diagnostic specification).

There are two possible ways to send an extended negative response:

6.4.1 Sending an extended negative response during service processing

If the application is currently processing a request which requires an extended negative response, the standard function **DescSetNegResponse(errorCode)** (please refer to the general technical reference document “TechnicalReference_CANdesc” for more details) cannot be used. Instead, the following function is defined:

Prototype	
Single Context	

`void DescSetExtNegResponse (DescNegResCode errorCode,`
`                             DescExtNegResCode  extErrorCode)`	

Multi Context	

`void DescSetExtNegResponse (vuint8 iContext,`
`                             DescNegResCode errorCode,`
`                             DescExtNegResCode  extErrorCode)`	

Parameter	
`iContext`	Reference to the corresponding request context
`errorCode`	One of the error code constants defined by CANdesc (located in the generated desc.h file) with the following naming convention: **kDescNrc<error name>**. Normally, only error code 0xE3 is used.
`extErrorCode`	A two byte value which is ECU/use-case dependent
Return code	
–	-
Functional Description	
In the pre-handler or main-handler callback, the application can call this function to send an extended negative response. Normally, this extended negative response is only useful for service $AE, but the function may be used for any currently active service (prior to calling **DescProcessingDone()**).	
Particularities and Limitations	
▶ The application must already have initialized CANdesc by calling **DescInitPowerOn()** ▶ The define DESC_ENABLE_EXT_NEG_RES_CODE_HANDLING must exist in the generated code ▶ Once an error code has been set it cannot be overwritten or reset. ▶ This function does not finish the processing of the request. The application must confirm that the request processing is completely finished by calling **DescProcessingDone()**.	
Call context	
▶ Within a service pre-handler callback and within or after a service main-handler callback	

Table 6-3 DescSetExtNegResponse

6.4.2 Sending an unsolicited extended negative response

If the application is not currently processing a request but an extended negative response must be sent, the function above cannot be used. Instead, a generic API for transmitting an unsolicited response can be used. In this case, the application must compose the response message on its own. The following is an example using the format description from the beginning of section 6.4:

```
/** Global buffer.
 */
static vuint8 applExtNrcMsgBuffer[5];

/** Initialize the const values.
 */
void ApplInit(void)
{
  /* Header for negative response. */
  applExtNrcMsgBuffer[0] = 0x7F;
  /* Only this service may use this functionality. */
  applExtNrcMsgBuffer[1] = 0xAE;
  /* Use NRC= 0xE3 */
  applExtNrcMsgBuffer[2] = kDescNrcDeviceControlLimitExceeded;

  /* The ext code will be set later.
  applExtNrcMsgBuffer[3] = ...
  applExtNrcMsgBuffer[4] = ...
  */
}

/** Shall be called when the device control
 *  out of limits is detected.
 */
void ApplSendUnsolicitedExtNrc(vuint16 extCode)
{
  applExtNrcMsgBuffer[3] = DescGetHiByte(extCode);
  applExtNrcMsgBuffer[4] = DescGetLoByte(extCode);

  /* Sent the message */
  DescTransmitSingleFrame(applExtNrcMsgBuffer, 5);
}
```

Figure 6-2 Example code for transmitting an unsolicited response

6.5 Sending an unsolicited single frame response

If service “DeviceControl” ($AE) has been activated and the application detects that conditions have changed detrimentally (e.g. they are “out of limits”) since service activation, GM/Opel requires that an unsolicited extended negative response shall be sent by the ECU. To accomplish this, the following API is provided which allows the ECU developer to send any single frame message using the physically addressed diagnostic response CAN ID. This API is not just a simple “re-transmitter”, calling the TP with the application data, but it also synchronizes the request with the current CANdesc reception/transmission state machine:

- If CANdesc is currently receiving a request, the unsolicited response will be delayed until the reception finishes (either with success or failure).

- If CANdesc is currently transmitting a response, the unsolicited response will be delay until the transmission finishes (either with success or failure).

See Table 6-4 DescTransmitSingleFrame for the API description.

Prototype	
`void` **`DescTransmitSingleFrame`**`(DescMsg resData, vuint8 resLen)`	
Parameter	
`resData`	Pointer to the application data
`resLen`	The length of the data (in bytes) to be sent
Return code	
–	-
Functional Description	
This function is called by CANdesc to send the unsolicited positive response on a tester present timeout or by the application to send an unsolicited extended negative response.	
Particularities and Limitations	
▶ The application must already have initialized CANdesc by calling **DescInitPowerOn().** ▶ The data pointed to by `resData` is not cached (copied to another buffer) by CANdesc, so the ECU developer should be careful not to use automatic variable references (non-static function local variables). ▶ The length of the data may not exceed the transport layer single frame length (seven bytes in the case of normal addressing and six bytes in the case of extended addressing).	
Call context	
▶ Background-loop level task context	

Table 6-4 DescTransmitSingleFrame

6.6 GM/Opel CANdesc state machine access

The states relevant to the programming sequence are modeled as a normal CANdesc state group. This also enables all the usual state group related callbacks and functions for transition notification, access to the current state, and a function to modify the current state. For naming convention and an API description, please refer to the general CANdesc technical reference.

The state group and its states' names have been fixed to provide a consistent API independent from the configuration. Normally the names would depend on the settings in the CDD file. Please refer to *Table 8-4 Programming Sequence state group* for the exact names.

For compatibility reasons CANdesc still provides the API described in this chapter although they be replaced by the state machine API.

6.6.1 Normal communication state

Prototype	
`vuint8 DescGetCommState (void)`	
Parameter	
-	-
Return code	
`kDescCommDisabled`	Normal message transmission is deactivated
`kDescCommEnabled`	Normal message transmission is activated
Functional Description	
The application can call this function at any time to obtain the current transmission state.	
Particularities and Limitations	
▶ The application must already have initialized CANdesc by calling **DescInitPowerOn()**. ▶ The same information can be retrieved using **DescGetStateProgrammingMode ()**. ▶ State information updates **after** any post handler of mode $28.	
Call context	
▶ Any	

Table 6-5 DescGetCommState

6.6.2 Programming mode state

Prototype	
`vuint8 DescGetProgMode (void)`	
Parameter	
–	-
Return code	
`kDescProgModeIdle`	No enter programming mode request up to now
`kDescProgModeAccepted`	Enter programming mode accepted (but not active yet)
`kDescProgModeActive`	Enter programming mode sequence complete
Functional Description	
The application can call this function at any time to read the “enter programming mode” sequence state.	
Particularities and Limitations	
▶ The application must already have initialized CANdesc by calling **DescInitPowerOn()**. ▶ The same information can be retrieved using **DescGetStateProgrammingMode ().** ▶ State information updates **after** any post handler of mode $A5.	
Call context	
▶ Any	

Table 6-6 DescGetProgMode

6.6.3 High speed programming mode state

Prototype	
Single channel	
`vuint8 DescGetHiSpeedMode (void)`	
Multiple channel	
`vuint8 DescGetHiSpeedMode (vuint8 commChannel)`	
Parameter	
`commChannel`	Communication channel on which CANdesc is running
Return code	
`kDescHiSpeedModeIdle`	No enter programming mode request up to now.
`kDescHiSpeedModeAccepted`	Enter high speed programming mode accepted (but not active yet)
`kDescHiSpeedModeActive`	Enter high speed programming mode sequence complete
Functional Description	
This function can be called by the application at any time to see if the programming mode requires a switch to high speed mode. If the define DESC_ENABLE_FLASHABLE_ECU exists in the generated code, then the application should call this function within the callback *ApplDescOnEnterProgMode* to decide whether to switch into high speed mode.	
Particularities and Limitations	
▶ The result is only valid for the channel on which CANdesc is running. ▶ The application has already initialized CANdesc once by calling **DescInitPowerOn().** ▶ The define DESC_ENABLE_REQ_HISPEED_PROG must exist in the generated code (if the ECU must support service $A5 $02). ▶ `Idle` and `Accepted` can be retrieved using **DescGetStateProgrammingMode ()** ▶ `Active` can be queried using IlNwmGetState() ▶ State information updates after any post handler of mode $A5.	
Call context	
▶ Any	

Table 6-7 DescGetHiSpeedMode

6.7 The PacketHandler (another type of service processor)

A main-handler is the typical callback function for request processing (please refer to the general technical reference document “TechnicalReference_CANdesc” for more details). For the GM/Opel version of CANdesc, a different type of service processor for *Service ReadDataByPacketIdentifier ($AA)* is necessary – the PacketHandler.

A PacketHandler is a very simple callback that has only one task – to query the application data and place it into the correct position of the response data buffer. The application may not use the negative response API with a PackedHandler. The API works asynchrounously, to allow moving time-consuming operations to different task contexts.

.

6.7.1 PacketHandler API

Prototype	
`void` **`ApplDescReadPack<Instance-Qualifier>`** `(DescMsg pMsg)`	
Parameter	
`pMsg`	A pointer to a buffer where the application must copy its data
Return code	
–	-
Functional Description	
CANdesc calls this function to query the application for the response data related to <Instance-Qualifier>. The application can provide the data within this function, or it can exit the function and provide it later on the task level. When the data is ready, the application must call **DescDataPacketProcessingDone()**. Once the application calls **DescDataPacketProcessingDone()**, the PacketHandler assembles the data to be sent with the response (the response length is predefined in CANdelaStudio by the DPID data structure definition).	
Particularities and Limitations	
▶ The application may not call **DescProcessingDone()**.	
Call context	
▶ Background-loop level context of DescStateTask().	

Table 6-8 PacketHandler

Prototype	
`void` **`DescDataPacketProcessingDone`** `(DescDataPacketProcessStatus status)`	
Parameter	
`status`	Valid values: ▶ `kDescDataPacketOk` – if the application copied the data successfully ▶ `kDescDataPacketFailedSendDummy` – if the application encountered an error while obtaining the requested data. CANdesc will transmit the UUDT response, but it will contain only the message padding pattern and no actual data. ▶ `kDescDataPacketFailedDoNotSend` - if the application encountered an error while obtaining the requested data. No UUDT response will be sent.
Return code	
–	-
Functional Description	
The application must call this function when it has finished the **ApplDescReadPack<Instance-Qualifier> (DescMsg pMsg)** request. This function allows the ECU developer to have a very slow PacketHandler (e.g. read data from another CPU). The CANdesc scheduler can process only one DPID at a time, so once the data has been copied to the pMsg buffer the application should call this function immediately to avoid long scheduler delays (time jitter from the configured scheduler rates).	
Particularities and Limitations	
▶ CANdesc must have previously called a PacketHandler callback **ApplDescReadPack<Instance-Qualifier> (DescMsg pMsg)**.	
Call context	
▶ Background-loop level context of DescStateTask().	

Table 6-9 DescDataPacketProcessingDone

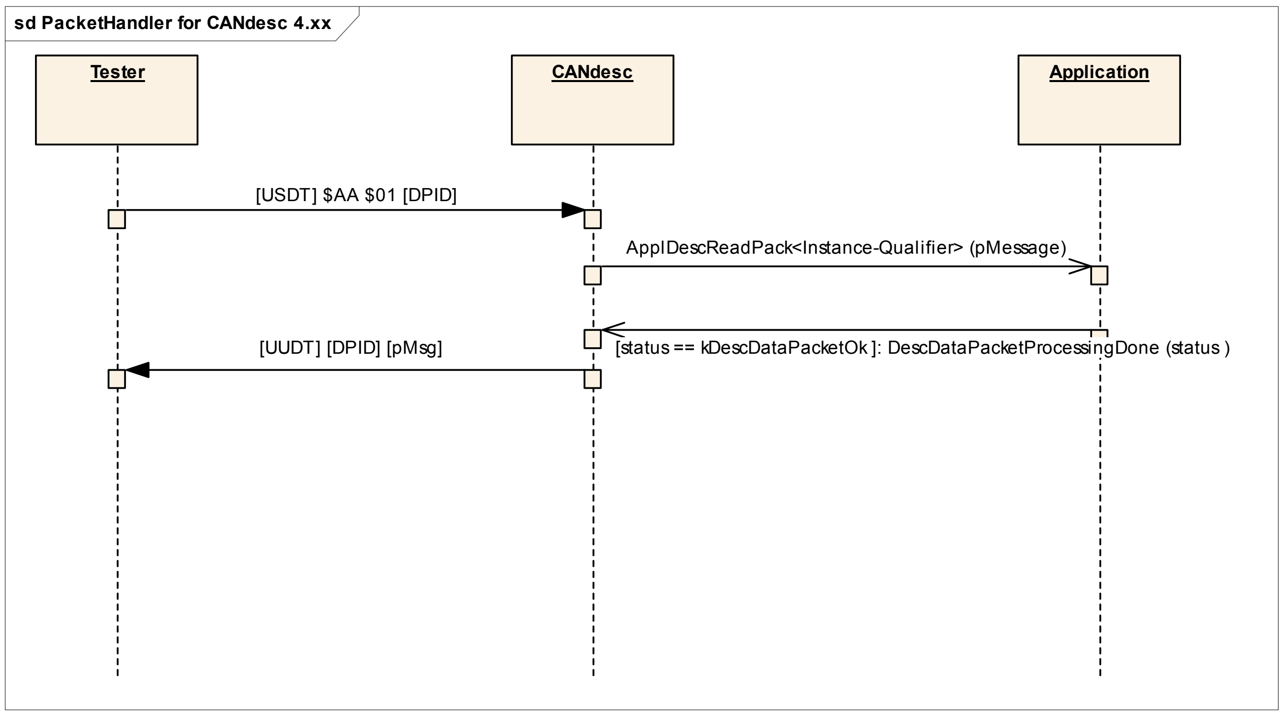

sd PacketHandler for CANdesc 4.xx
Tester
CANdesc
Application
[USDT] $AA $01 [DPID]
ApplDescReadPack<Instance-Qualifier> (pMessage)
[status == kDescDataPacketOk]: DescDataPacketProcessingDone (status)
[UUDT] [DPID] [pMsg]

Figure 6-3 Asynchronous PacketHandler for current CANdesc versions

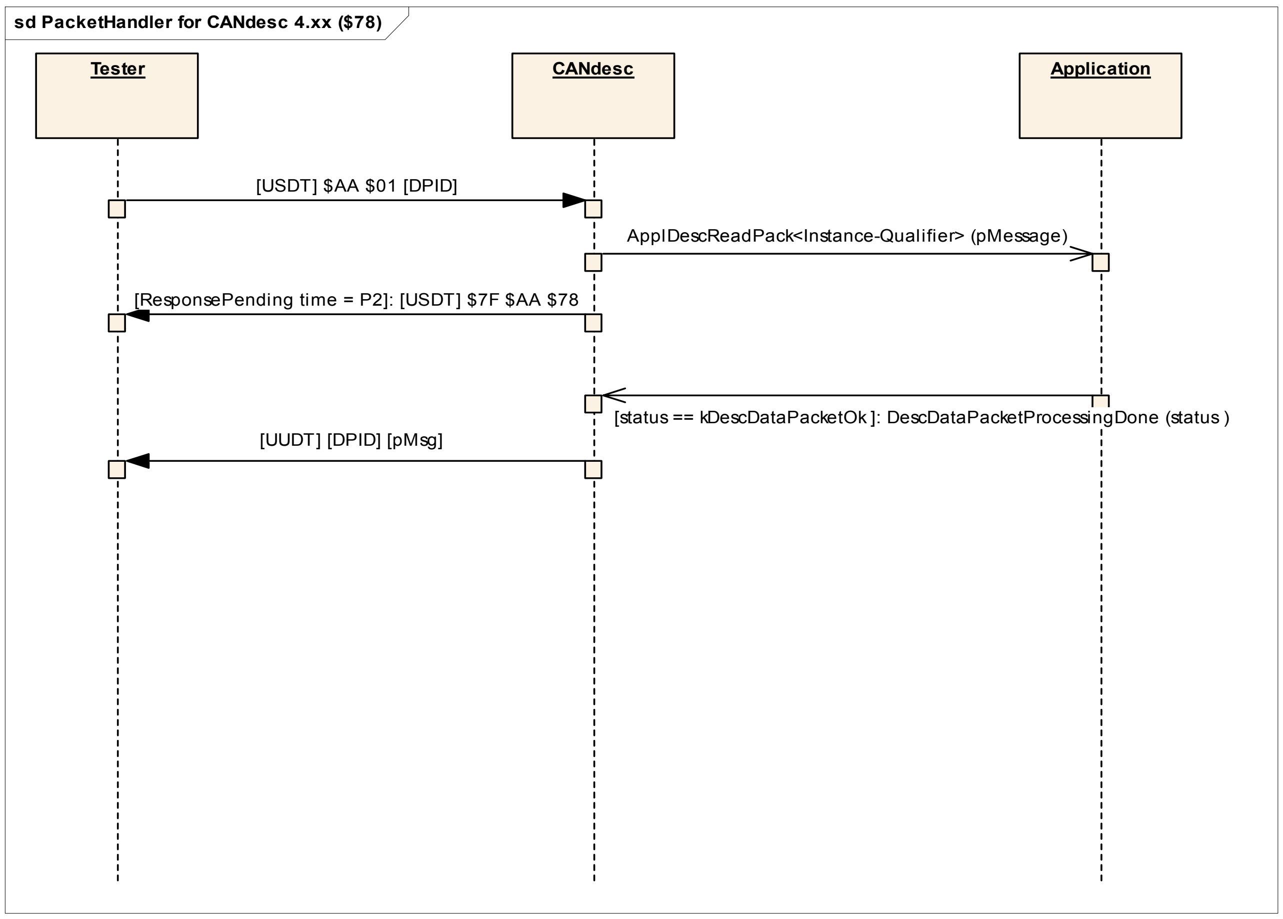

sd PacketHandler for CANdesc 4.xx ($78)
Tester
CANdesc
Application
[USDT] $AA $01 [DPID]
ApplDescReadPack<Instance-Qualifier> (pMessage)
[ResponsePending time = P2]: [USDT] $7F $AA $78
[status == kDescDataPacketOk]: DescDataPacketProcessingDone (status)
[UUDT] [DPID] [pMsg]

Figure 6-4 Asynchronous PacketHandler with delayed processing of the first data packet

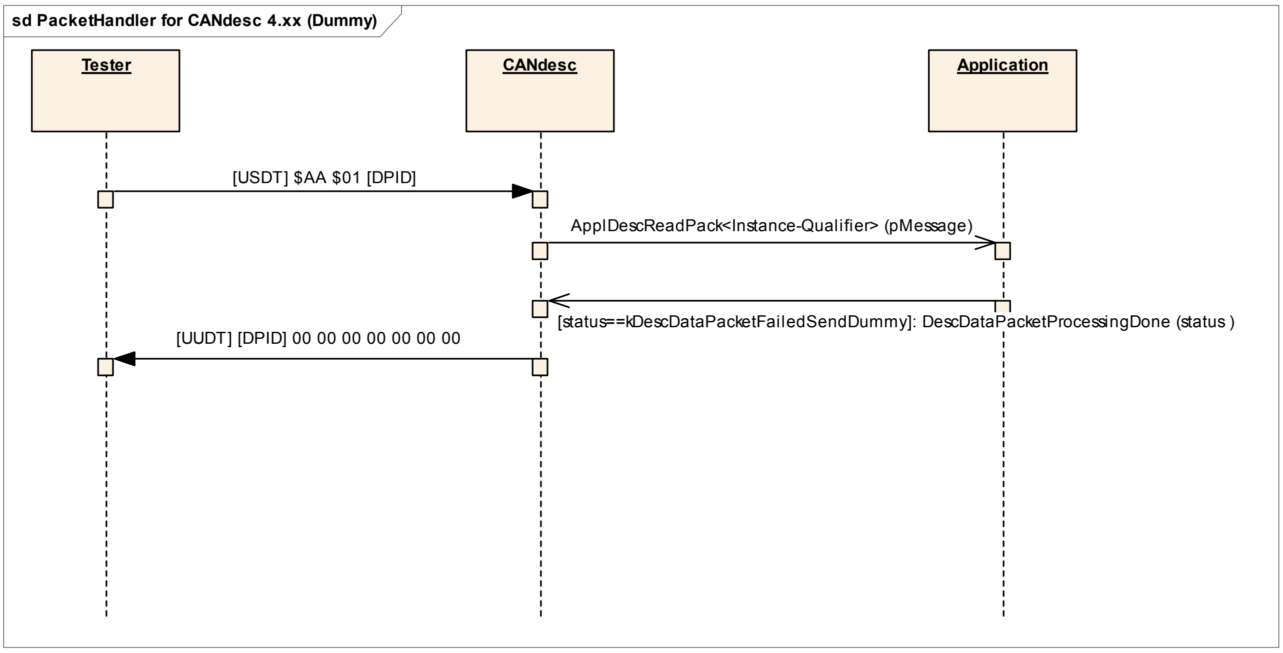

sd PacketHandler for CANdesc 4.xx (Dummy)
Tester
CANdesc
Application
[USDT] $AA $01 [DPID]
ApplDescReadPack<Instance-Qualifier> (pMessage)
[status==kDescDataPacketFailedSendDummy]: DescDataPacketProcessingDone (status)
[UUDT] [DPID] 00 00 00 00 00 00 00

Figure 6-5 Asynchronous PacketHandler with dummy response, due to an application error while obtaining response data

7 GM/Opel service implementations

7.1 Service InitiateDiagnosticOperation ($10)

CANdesc implements a special handling of the following sub-functions of this service:

- "DisableAllDtcs" (sub-function $02)
- "EnableDtcsDuringDeviceControl" (sub-function $03)

WakeUpLinks (sub-function $04) can be implemented using standard main- and post-handlers.

Note
The actual implementation of these services is still left to the application, controlled by the callbacks mentioned below. CANdesc only manages the internal states and performs service validation.

7.1.1 Service DisableAllDTCs ($10 $02)

Prototype	
`void ApplDescOnDisableAllDtc (void)`	
Parameter	
–	-
Return code	
–	-
Functional Description	
Once the service $10 $02 execution has completed with success, CANdesc calls this function to notify the application about the new state of the diagnostics module.	
Particularities and Limitations	
▶ In this callback the application must implement the actual disabling of DTCs. ▶ Also refer to *ApplDescOnReturnToNormalMode* in order to re-enable DTCs.	
Call context	
▶ Background-loop level task context of DescStateTask()	

Table 7-1 ApplDescOnDisableAllDtc

7.1.2 Service EnableDTCsDuringDeviceControl ($10 $03)

Prototype	
`void` **`ApplDescOnEnableDtcsChangeDuringDevCntrl`** `(void)`	
Parameter	
–	-
Return code	
–	-
Functional Description	
Once the service $10 $03 execution has completed with success, CANdesc calls this function to notify the application about the new state of the diagnostics module.	
Particularities and Limitations	
▶ In this callback, the application must enable DTCs during device control. ▶ Also refer to *ApplDescOnReturnToNormalMode.*	
Call context	
> Background-loop level task context of DescStateTask().	

Table 7-2 ApplDescOnEnableDtcChangeDuringDevCntrl

Note
CANdesc activates the tester present timer for both service instances above.

7.2 Service ReadFailureRecordData ($12)

CANdesc does not offer any special support for this service, but since the definition of the service is not compatible with the dispatching ability of CANdesc, there are limitations on the application design:

7.2.1 Service ReadFailureRecordIdentifiers ($12 $01)

Request

CANdesc fully dispatches the diagnostic request; the application does not have to perform validation.

Response

Depending on the report type requested (PID or DPID), the application must place one of the following values into the first data byte of the response message:

- 0x00 - for report by PID
- 0x01 - for report by DPID

Note
The ECU can support either reports by PID or DPID, but not both.

7.2.2 Service ReadFailureRecordParameters ($12 $02)

Request

The diagnostic request $12 $02 uses a PID parameter, which is not automatically dispatched by CANdesc. As a result, CANdesc generates only one function callback (main-handler) for all service $12 $02 requests. The application must validate the PID parameter, check the request length, and then process the PID accordingly.

Response

CANdesc does not automatically include the PID parameter in the response message for service $12 $02. The application must perform this task.

7.3 Service ReturnToNormalMode ($20)

CANdesc completely handles this service and performs the following actions after a positive response has been sent (or after finishing service execution when the request does not require a response):

- The tester present timer will be deactivated.
- If communication was disabled, it will be re-enabled. CANdesc will notify the application via the callback function *ApplDescOnEnableNormalComm*.
- The network management timer will be started by a new request (timeout: 8 seconds).
- If the service *SendOnChangeDTCCount ($A9 $82)* was active, CANdesc will notify the application to deactivate it by calling the function *ApplDescDisableOnChangeDtcCount*.
- The scheduling mechanism for the *Service ReadDataByPacketIdentifier ($AA)* will be deactivated, and the scheduler lists will be cleared.

CANdesc also notifies the application of the return to normal mode event via the function call *ApplDescOnReturnToNormalMode*:

Prototype	
`void` **`ApplDescOnReturnToNormalMode`** `(void)`	
Parameter	
–	-
Return code	
–	-
Functional Description	
CANdesc calls this function on completion of service $20 processing (after the response has been sent, if applicable) or a tester present timeout.	
Particularities and Limitations	
▶ None	
Call context	
▶ Background-loop level task context of DescStateTask().	

Table 7-3 ApplDescOnReturnToNormalMode

7.4 Service ReadDataByParameterIdentifier ($22)

The description for this service is located in the general technical reference document “TechnicalReference_CANdesc”. Detailed below are only the deviations and behavior not defined in the general technical reference.

7.4.1 Reading a dynamically defined PID (Parameter Identifier)

Some PIDs are statically defined in CANdelaStudio (i.e. their data structure is predefined at compile time); however, others can only be defined at runtime using a special service request (see

Service DefinePIDByAddress ($2D). These dynamically definable PIDs have no data definitions at power-on, so if the tester tries to read them the result is undefined (according to GMW-3110 version 1.6). In this situation, CANdesc sends a positive response with no actual data (only the response service ID ($62) and the PID number from the request).

Please refer to the sequence in *Figure 7-9 The dynamically definable PID is not defined.*

7.5 Service SecurityAccess ($27)

In general, the application shall implement this service by itself, but CANdesc already handles some of the tasks regarding this implementation. In the following, you will learn about what already has been done for you by CANdesc and about the remained parts to be implemented by your application.

CANdesc tasks:

- Service format verification.
  This step includes: request length and supported sub-service checks.
- Sub-service execution preconditions specified in the CDD file.
- SecurityAccess state change
  according to the CDD file on positive response for “send-key” service(s).

Application tasks:

- Additional preconditions checks (pre-handling).
- Seed-key sequence verification.
- Timeout monitoring for next seed retry.
- MEC and vulnerability flag implementation (if applicable).
- Seed generation.
- Key computation and verification upon requested key.
- In case of valid keys, starting of the TesterPresent timer in CANdesc (see API DescActivateS1Timer). The best place to do that would be the post-handler of a “send-key” service. Just verify if the service has been responded positively, and the response was sent successfully. For details about service post-handling, please refer the OEM independent CANdesc TechnicalReference file located in the delivered software package.

7.6 Service DisableNormalCommunication ($28)

The GM/Opel version of CANdesc takes the following actions prior sending the positive response:

- Requests GMLAN/IVLAN to disable normal communication
- If the disable normal communication request succeeds:
  - Sets the new communication status (*kDescCommDisabled*)
  - Activates the tester present timer
  - Notifies the application via the function call *ApplDescOnDisableNormalComm*.
  - Positive response will be sent
- If the disable normal communication request fails:
  - A negative response with NRC $22 (ConditionsNotCorrect) will be sent

Prototype	
`void ApplDescOnDisableNormalComm (void)`	
Parameter	
–	-
Return code	
–	-
Functional Description	
CANdesc calls this function once normal message transmission has been disabled by a service $28 request and the resulting positive response has been sent.	
Particularities and Limitations	
▶ None	
Call context	
▶ Background-loop level task context of DescStateTask().	

Table 7-4 ApplDescOnDisableNormalComm

Prototype	
`void ApplDescOnEnableNormalComm (void)`	
Parameter	
–	-
Return code	
–	-
Functional Description	
Once normal message transmission has been restored, CANdesc calls this function to notify the application.	
Particularities and Limitations	
▶ None	
Call context	
▶ Background-loop level task context of DescStateTask().	

Table 7-5 ApplDescOnEnableNormalComm

7.7 Service DynamicallyDefineMessage ($2C)

The GM diagnostic specification (GMW-3110 version 1.6) allows some DPIDs to be defined at runtime by the tester, rather than at compile time. Using this technique, the tester can map one or more PIDs (statically or dynamically defined) to a DPID and then read them back via *Service ReadDataByPacketIdentifier ($AA)*.

CANdesc completely implements this service; no application implementation is required.

The diagram below depicts the relationship between statically and dynamically defined DPIDs and PIDs and the services that can access them.

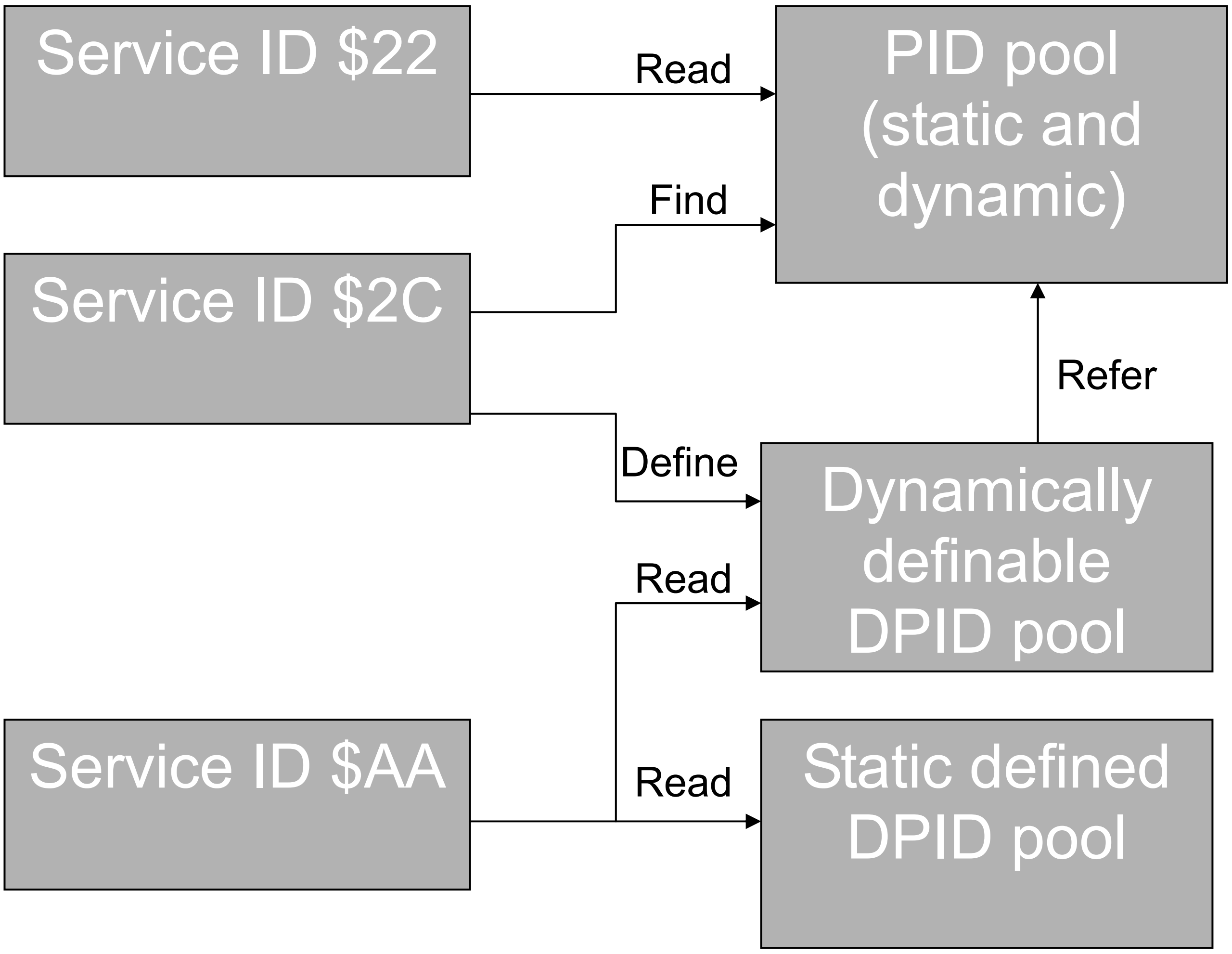


Figure 7-1 Dynamically defined DPID service relations

7.8 Operations on dynamically definable DPIDs

Dynamically definable DPIDs can be defined and read. The following sections illustrate these operations with sequence charts involving the tester, CANdesc, and application.

7.8.1 Defining a dynamically definable DPID

If the requested parameters are valid (both PIDs referenced and DPID to be defined are valid, request has the correct length, etc.), CANdesc overwrites the current DPID definition with the new list of PIDs.

Caution
If the tester request contains multiple instances of the same PID, the resulting DPID definition will contain multiple instances as well. CANdesc does not verify that all of the requested PIDs are unique.

7.8.2 Reading a dynamically definable DPID

When a read operation is performed on a dynamically definable DPID, four scenarios are possible:

- The DPID is not defined
- The DPID is defined, but the data is not accessible at the moment
- The DPID is defined, and the data is accessible
- The DPID is defined, but one of the contained PIDs is dynamically definable and it has not been defined

If the DPID is not defined, the CANdesc scheduler will send UUDT messages with no actual data content (only zeros):

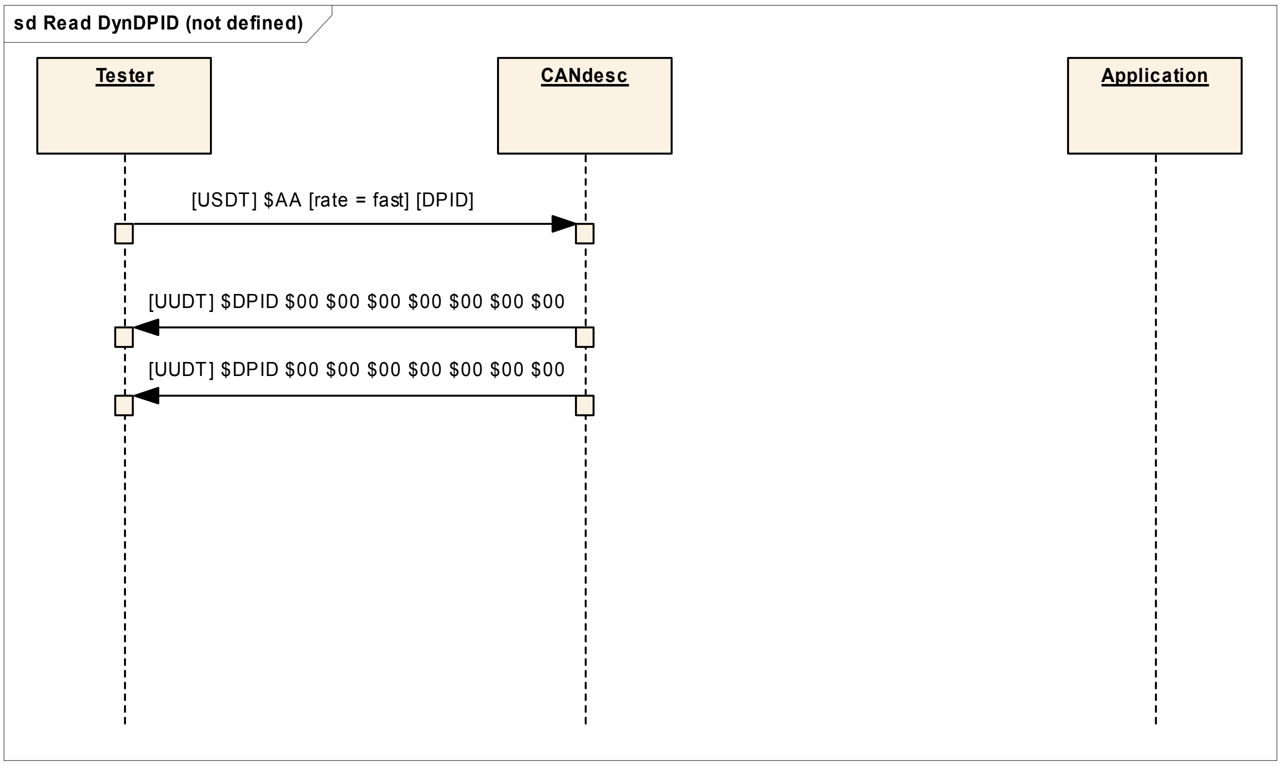

sd Read DynDPID (not defined)
Tester
CANdesc
Application
[USDT] $AA [rate = fast] [DPID]
[UUDT] $DPID $00 $00 $00 $00 $00 $00 $00
[UUDT] $DPID $00 $00 $00 $00 $00 $00 $00

Figure 7-2 Dynamically definable DPID is not defined

The next example assumes that the dynamically definable DPID has already been defined and references a single PID. In addition, the reading of this PID is not possible at the time of data access (e.g. the data is corrupted), and therefore the application would like to reject the PID processing. In this situation, the UUDT response message would once again contain no actual data (only zeros):

Caution
In the case where multiple PIDs are contained in the DPID, and some of them cannot be processed with success, these PIDs will be skipped in the UUDT response.

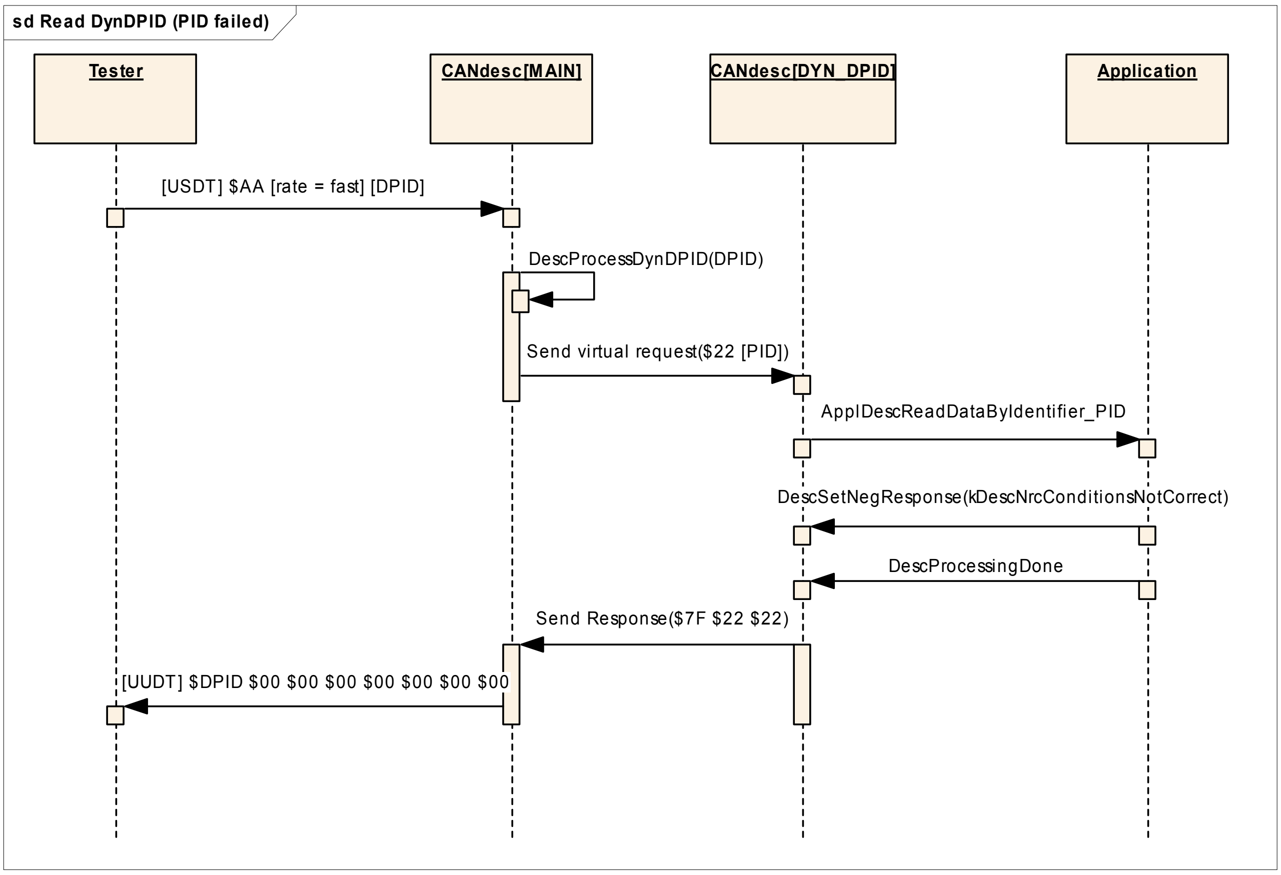

sd Read DynDPID (PID failed)
Tester
CANdesc[MAIN]
CANdesc[DYN_DPID]
Application
[USDT] $AA [rate = fast] [DPID]
DescProcessDynDPID(DPID)
Send virtual request($22 [PID])
ApplDescReadDataByIdentifier_PID
DescSetNegResponse(kDescNrcConditionsNotCorrect)
DescProcessingDone
Send Response($7F $22 $22)
[UUDT] $DPID $00 $00 $00 $00 $00 $00 $00

Figure 7-3 Single referenced PID cannot provide any data

If the PID is accessible at the time of the request, the UUDT message will contain its data:

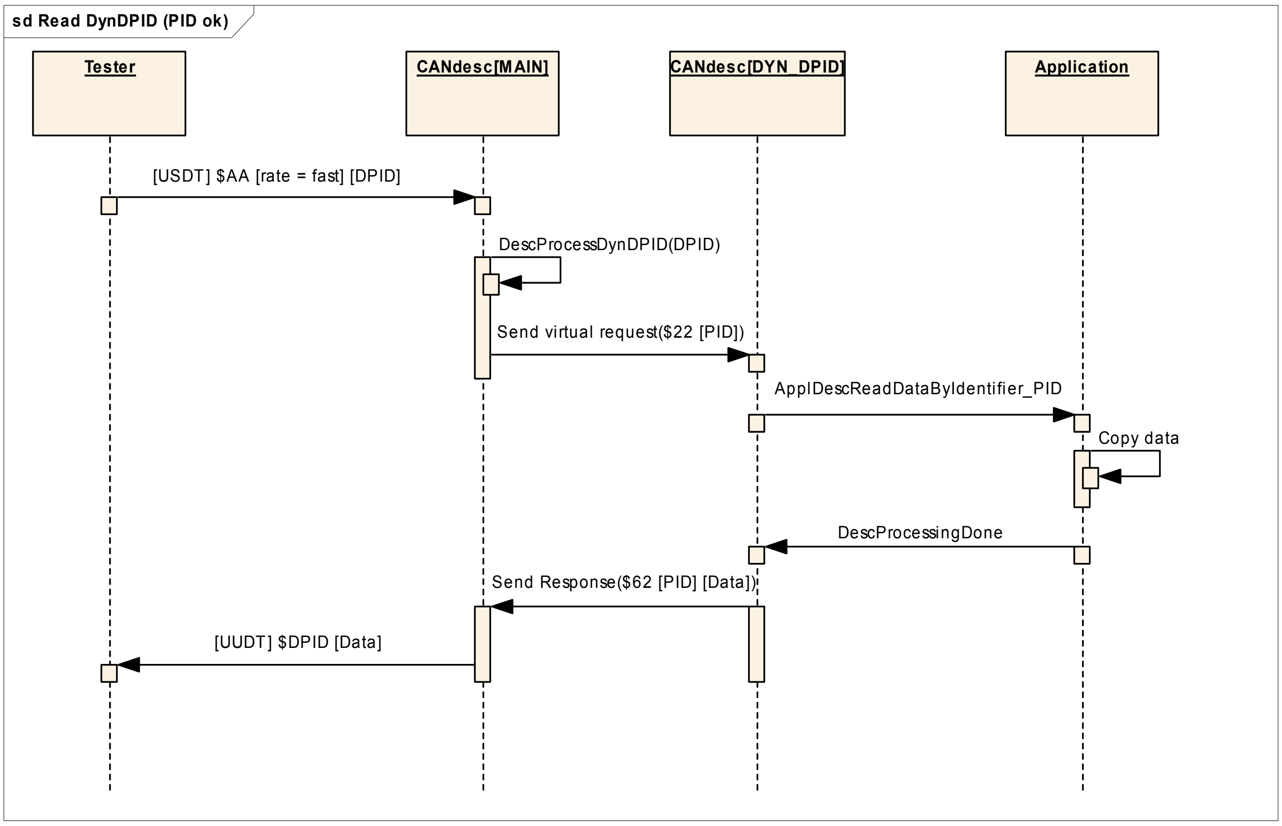


Figure 7-4 Reading the dynamically defined DPID succeeds

Since dynamically defined DPIDs can also contain dynamically definable PIDs, any dynamically definable PIDs within the requested DPID must be defined, or the resulting UUDT response will not contain any actual data (only zeros):

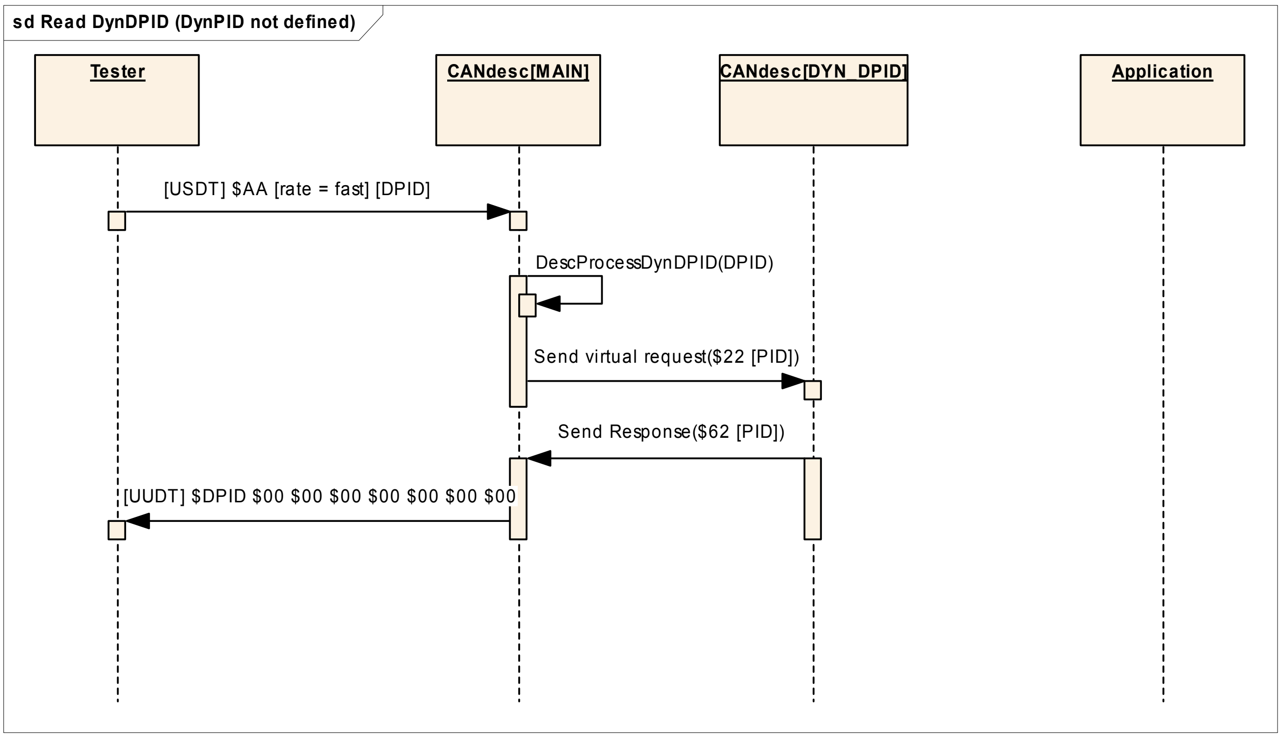

sd Read DynDPID (DynPID not defined)
Tester
CANdesc[MAIN]
CANdesc[DYN_DPID]
Application
[USDT] $AA [rate = fast] [DPID]
DescProcessDynDPID(DPID)
Send virtual request($22 [PID])
Send Response($62 [PID])
[UUDT] $DPID $00 $00 $00 $00 $00 $00 $00

Figure 7-5 Dynamically definable PID, referenced by the DPID, is not defined

Caution
When a read operation is performed on a DPID which contains a dynamically definable PID that is not defined but also other defined PIDs, the resulting UUDT response will not contain the data for this PID.

7.9 Service DefinePIDByAddress ($2D)

The GM diagnostic specification (GMW-3110 version 1.6) allows some PIDs to be defined at runtime by the tester, rather than at compile time. Using this technique, the tester can map a certain memory area to a PID and then read it back via *Service ReadDataByParameterIdentifier ($22)* or pack the PID into a DPID (*Service DynamicallyDefineMessage ($2C)*) for future read operations via *Service ReadDataByPacketIdentifier ($AA)*.

CANdesc completely implements this service. The application must only implement the memory area validation and access functionality.

The diagram below depicts the relationship between statically and dynamically defined PIDs and the services that can access them.

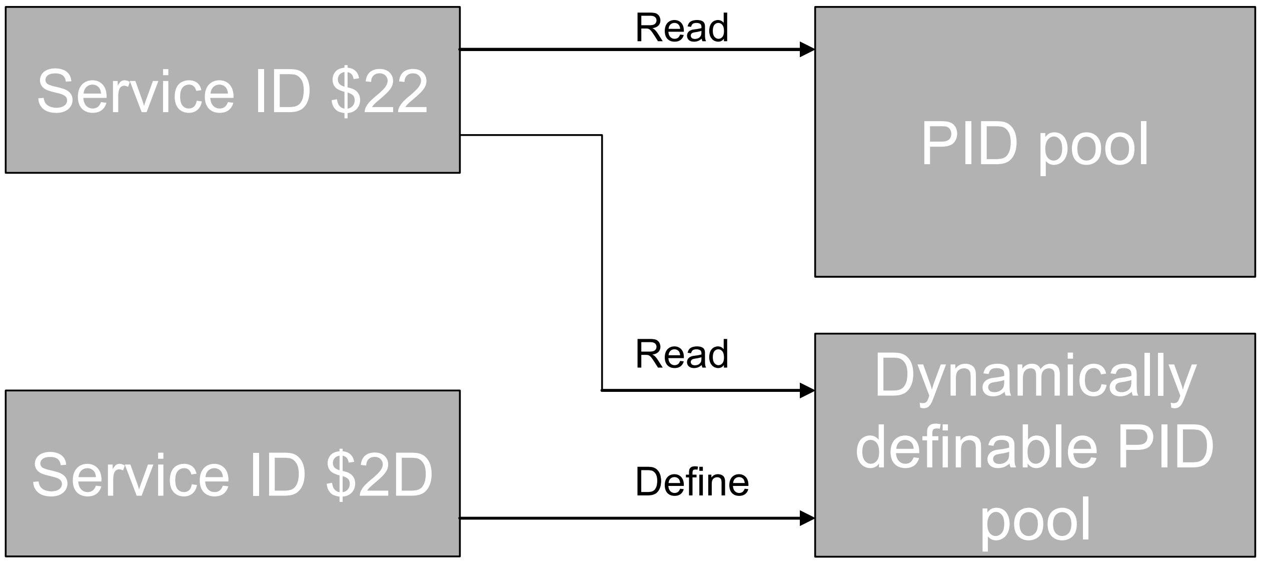


Figure 7-6 Dynamically defined PID service relations

7.10 Operations on dynamically definable PIDs

Dynamically definable PIDs can be defined and read. The following sections illustrate these operations with sequence charts involving the tester, CANdesc, and application.

7.10.1 Defining a dynamically definable PID

If the requested memory area parameters are valid and do not refer to a secured location, the following diagram applies:

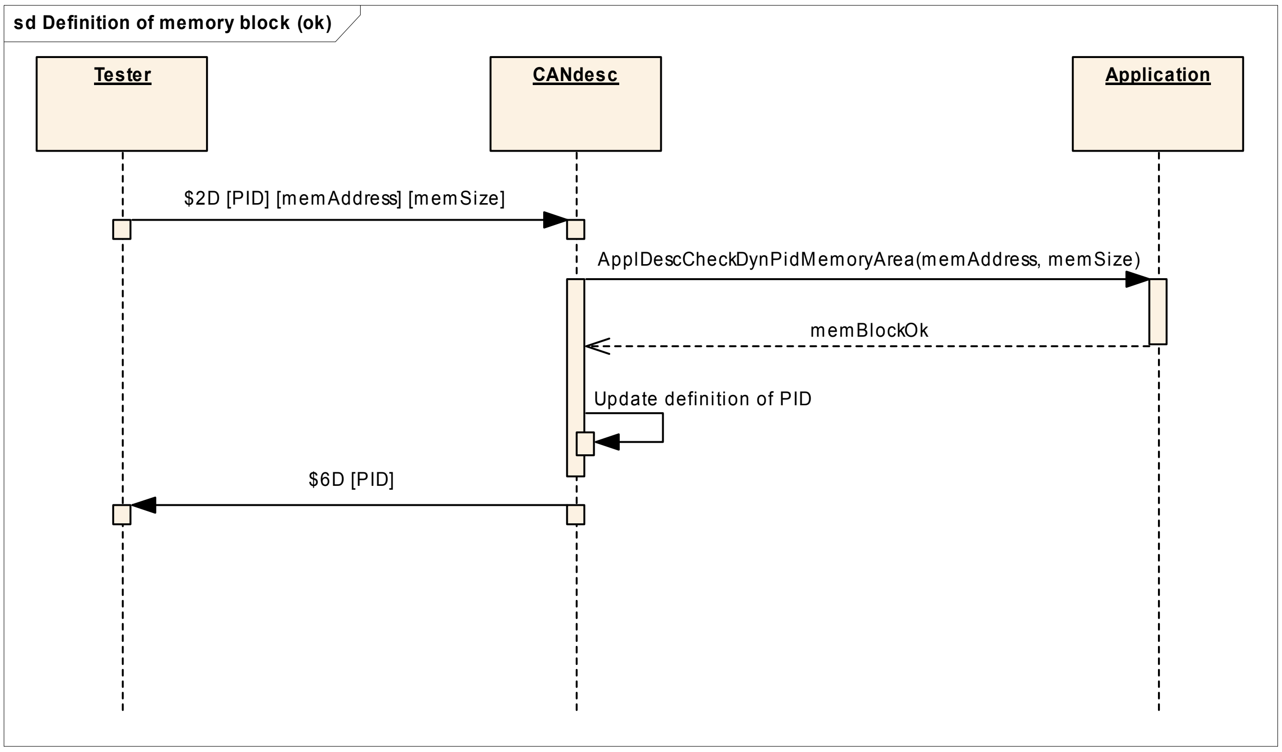

sd Definition of memory block (ok)
Tester
CANdesc
Application
$2D [PID] [memAddress] [memSize]
ApplDescCheckDynPidMemoryArea(memAddress, memSize)
memBlockOk
Update definition of PID
$6D [PID]

Figure 7-7 Definition of dynamically definable PID with valid parameter

If the application detects a problem with the requested memory area (e.g. bad location reference, security protection, etc.), the sequence below will take place:

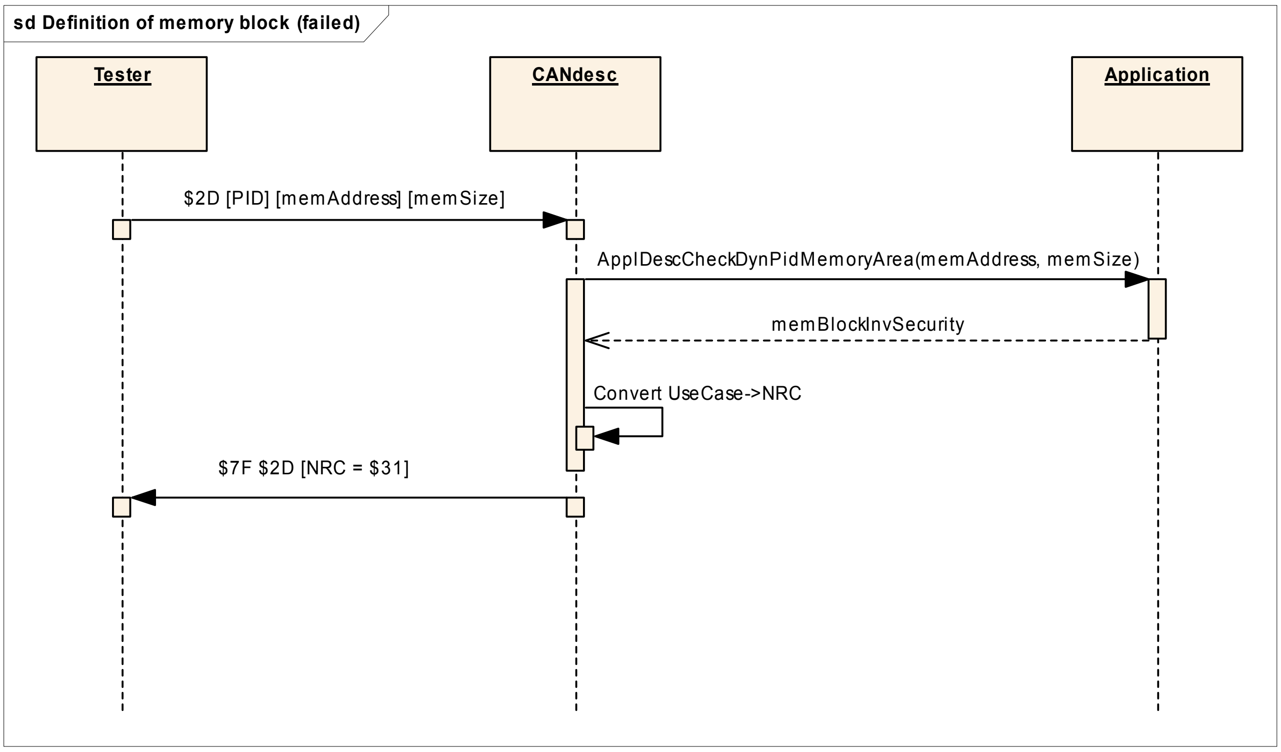

sd Definition of memory block (failed)
Tester
CANdesc
Application
$2D [PID] [memAddress] [memSize]
ApplDescCheckDynPidMemoryArea(memAddress, memSize)
memBlockInvSecurity
Convert UseCase->NRC
$7F $2D [NRC = $31]

Figure 7-8 Defining of a dynamically definable PID failed due to security violation

7.10.2 Reading a dynamically definable PID

When a read operation is performed on a dynamically definable PID, three scenarios are possible:

- The PID is not defined
- The PID is defined, but the data is not accessible at the moment
- The PID is defined, and the data is accessible

If the PID is not defined, CANdesc returns a positive response with no actual data bytes:

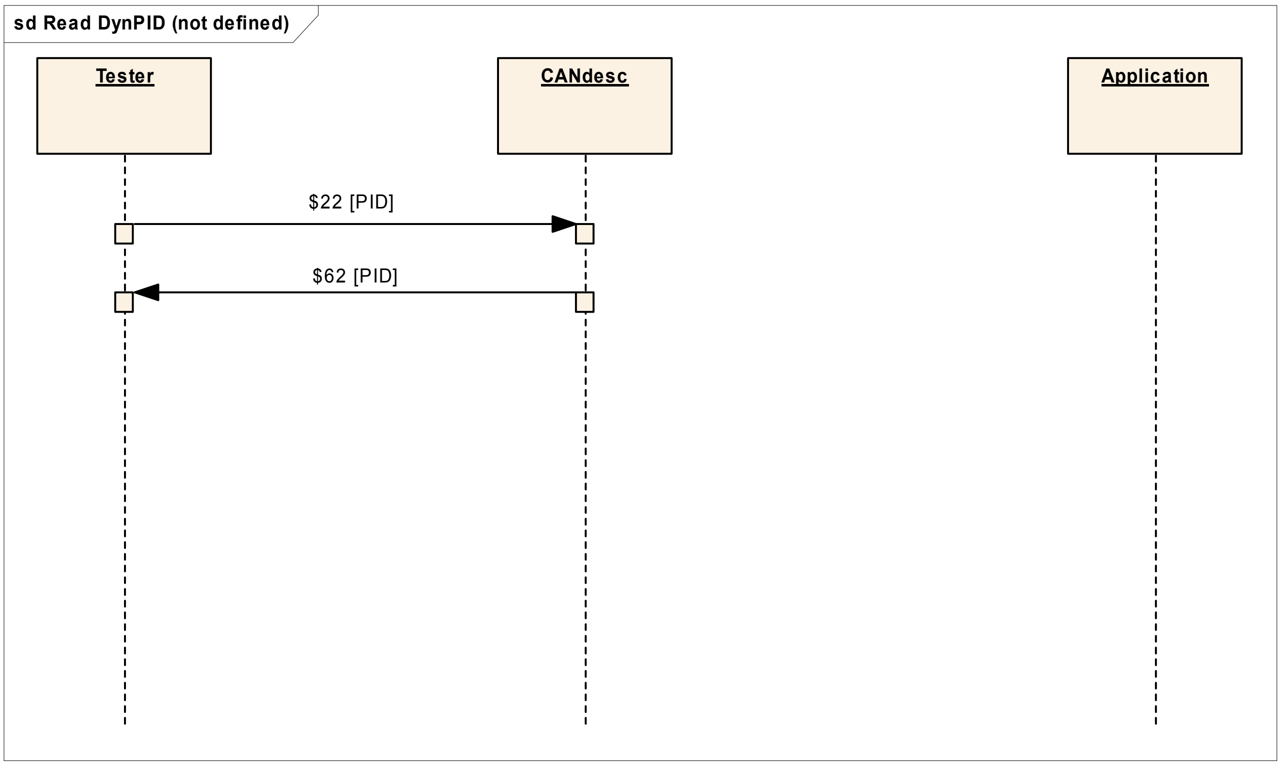


Figure 7-9 The dynamically definable PID is not defined

If the application has already accepted the PID read request (in *ApplDescCheckDynPidMemoryArea*) but it cannot complete the read operation, a negative response can still be sent back to the tester:

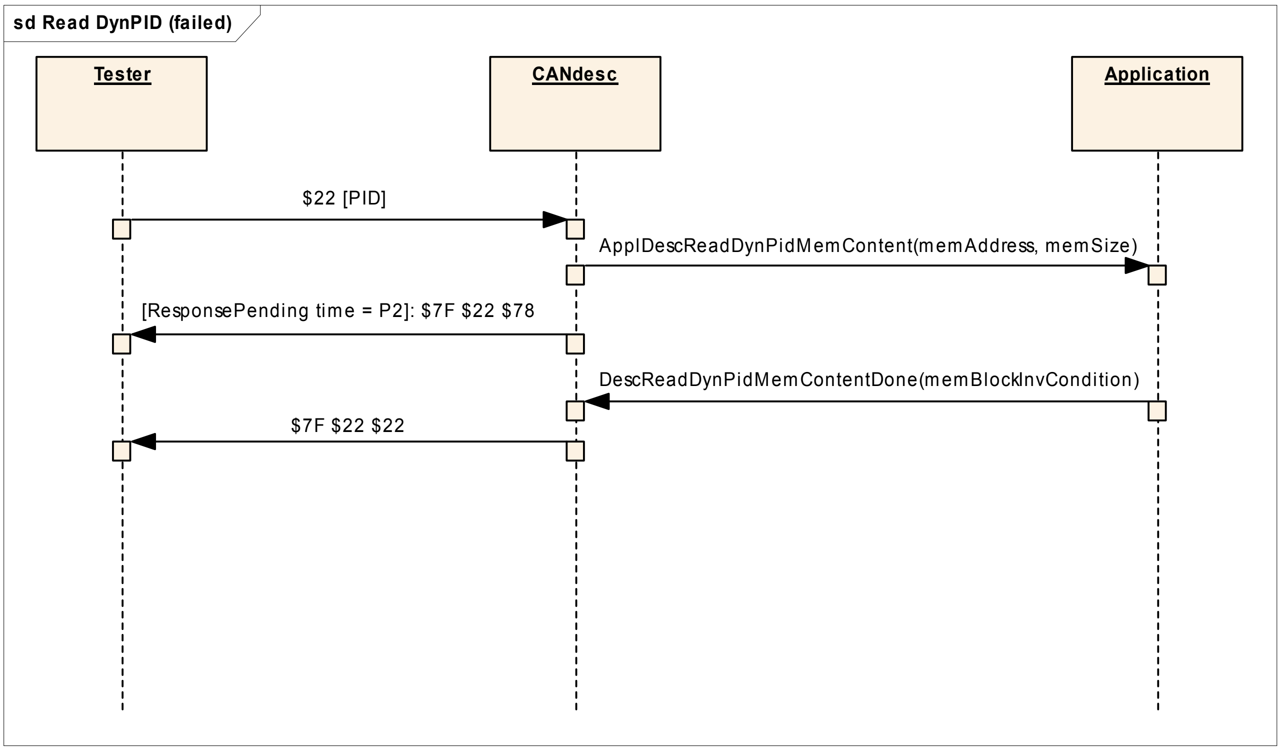

sd Read DynPID (failed)
Tester
CANdesc
Application
$22 [PID]
ApplDescReadDynPidMemContent(memAddress, memSize)
[ResponsePending time = P2]: $7F $22 $78
DescReadDynPidMemContentDone(memBlockInvCondition)
$7F $22 $22

Figure 7-10 Application error when reading the memory block

In the ideal case, the application writes the data into the supplied data buffer (see *ApplDescCheckDynPidMemoryArea*) and acknowledges the end of writing with success:

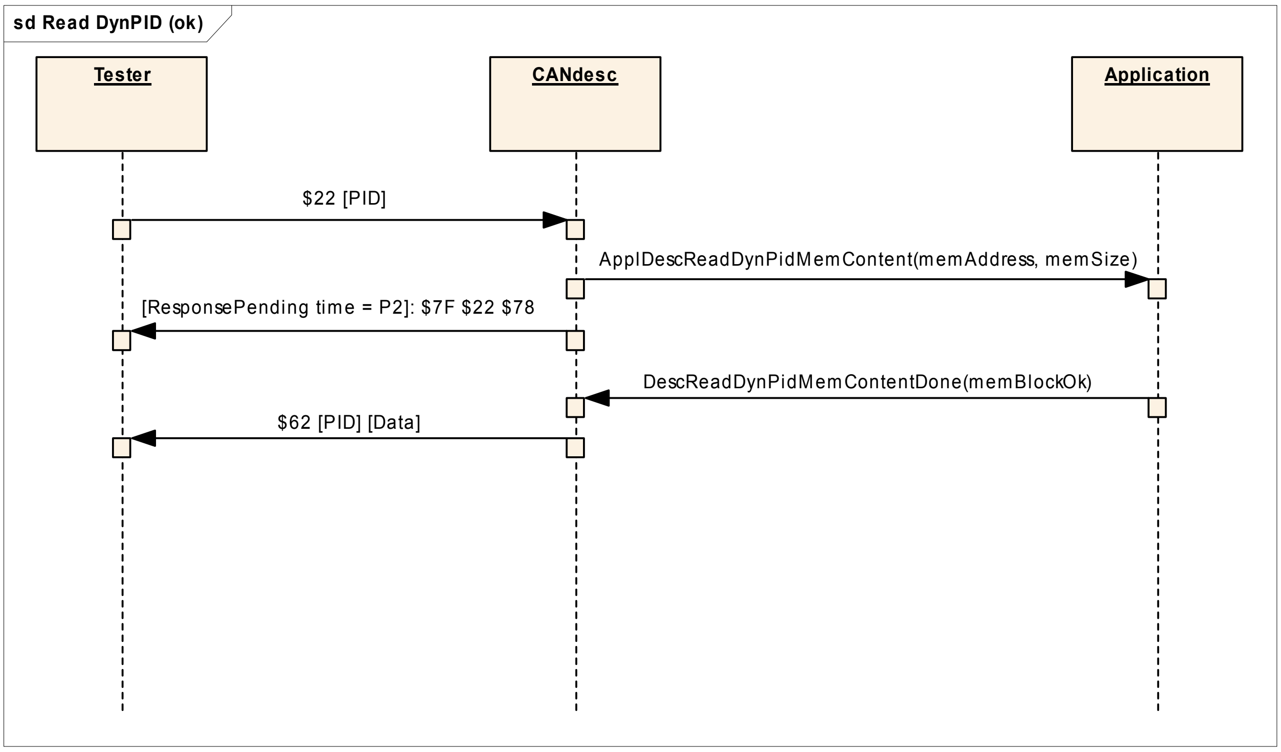

sd Read DynPID (ok)
Tester
CANdesc
Application
$22 [PID]
ApplDescReadDynPidMemContent(memAddress, memSize)
[ResponsePending time = P2]: $7F $22 $78
DescReadDynPidMemContentDone(memBlockOk)
$62 [PID] [Data]

Figure 7-11 Reading of dynamically defined PID succeeds

Prototype	
Single Context	
`DescDynPidMemAccessResult` **`ApplDescCheckDynPidMemoryArea`** `(const DescDynPidMemBlockInfo* const memArea)`	
Multi Context	
`DescDynPidMemAccessResult` **`ApplDescCheckDynPidMemoryArea`** `(vuint8 iContext, const DescDynPidMemBlockInfo* const memArea)`	
Parameter	
`iContext`	Current request handle (reserved for future use)
`memArea`	A pointer to the memory area definition which must be validated by the application
> `Address[]`	The requested start address (can be a 2, 3 or 4 byte array)
> `size`	The requested memory block size
Return code	
`memBlockOk`	The requested area is valid (no memory protection violation, range ok, etc.)
`memBlockInvAddress`	The requested memory address is invalid (wrong memory access)
`memBlockInvSize`	The requested memory size is invalid (e.g. out of range)
`memBlockInvSecurity`	The requested memory address and size are valid, but the area is protected by security
`memBlockInvCondition`	The access conditions are not correct
Functional Description	
CANdesc calls this function when the tester requests a valid $2D service (according to the specification). The application must validate the memory area parameters. When the application exits the function, CANdesc generates the appropriate NRC based on the application return value.	
Particularities and Limitations	
▶ Only available if the ECU supports service $2D. ▶ This function runs synchronously, so the application should exit the function as quickly as possible.	
Call context	
▶ Background-loop level task context of DescStateTask().	

Table 7-6 ApplDescCheckDynPidMemoryArea

Prototype	
Single Context	
`void **ApplDescReadDynPidMemContent** (DescMsg pMsg, const DescDynPidMemBlockInfo* const memArea)`	
Multi Context	
`void **ApplDescReadDynPidMemContent** (vuint8 iContext, DescMsg pMsg, const DescDynPidMemBlockInfo* const memArea)`	
Parameter	
`iContext`	Current request handle (reserved for future use)
`pMsg`	A pointer to a buffer where the application must write the requested memory contents
`memArea`	A pointer to the memory area definition which the application can use for data acquisition:
> `Address[]`	The requested start address (can be a 2, 3 or 4 byte array)
> `size`	The requested memory block size
Return code	
–	-
Functional Description	
Once defined, a dynamically definable PID can be read by the tester using service $22. Dynamically defined PIDs have special main-handlers which are implemented internally by CANdesc. These main-handlers call this function to retrieve the data at the requested memory address. CANdesc does not use a direct memory access implementation for this service since certain ranges may be located in slow (external) memory chips that require extended periods of time to access. As a result, this data acquisition function is designed to be asynchronous. Once the requested data bytes have been written into the buffer pointed to by pMsg, the application must call *DescReadDynPidMemContentDone* to acknowledge that the operation is complete.	
Particularities and Limitations	
> Only available if the ECU supports service $2D. > This function runs asynchronously, so once the application exits this function it can take as much time as necessary to complete the requested operation	
Call context	
> Background-loop level task context of DescStateTask().	

Table 7-7 ApplDescReadDynPidMemContent

Prototype	
Single Context	
`void DescReadDynPidMemContentDone (DescDynPidMemAccessResult result)`	
Multi Context	
`void DescReadDynPidMemContentDone (vuint8 iContext,` `DescDynPidMemAccessResult result)`	
Parameter	
`iContext`	The request handle previously passed as a parameter to the application in the callback *ApplDescReadDynPidMemContent.*
`result`	The result of the operation: ▶ `memBlockOk`: The requested area is valid (no memory protection violation, range ok, etc.) ▶ `memBlockInvAddress`: The requested memory address is not valid (bad memory access) ▶ `memBlockInvSize`: The requested memory size is invalid (e.g. out of range) ▶ `memBlockInvSecurity`: The requested memory address and size are valid, but the area is protected by security ▶ `memBlockInvCondition`: The access conditions are not correct
Return code	
–	-
Functional Description	
Once the data requested by *ApplDescReadDynPidMemContent* has been written into the buffer pointed to by pMsg, the application must call this function to acknowledge that the operation is complete.	
Particularities and Limitations	
▶ Only available if the ECU supports service $2D.	
Call context	
▶ Background-loop level task context of DescStateTask().	

Table 7-8 DescReadDynPidMemContentDone

7.11 Service ProgrammingMode ($A5)

This service is implemented completely using the CANdesc state management.

The programming sequence follows the state graph shown below:

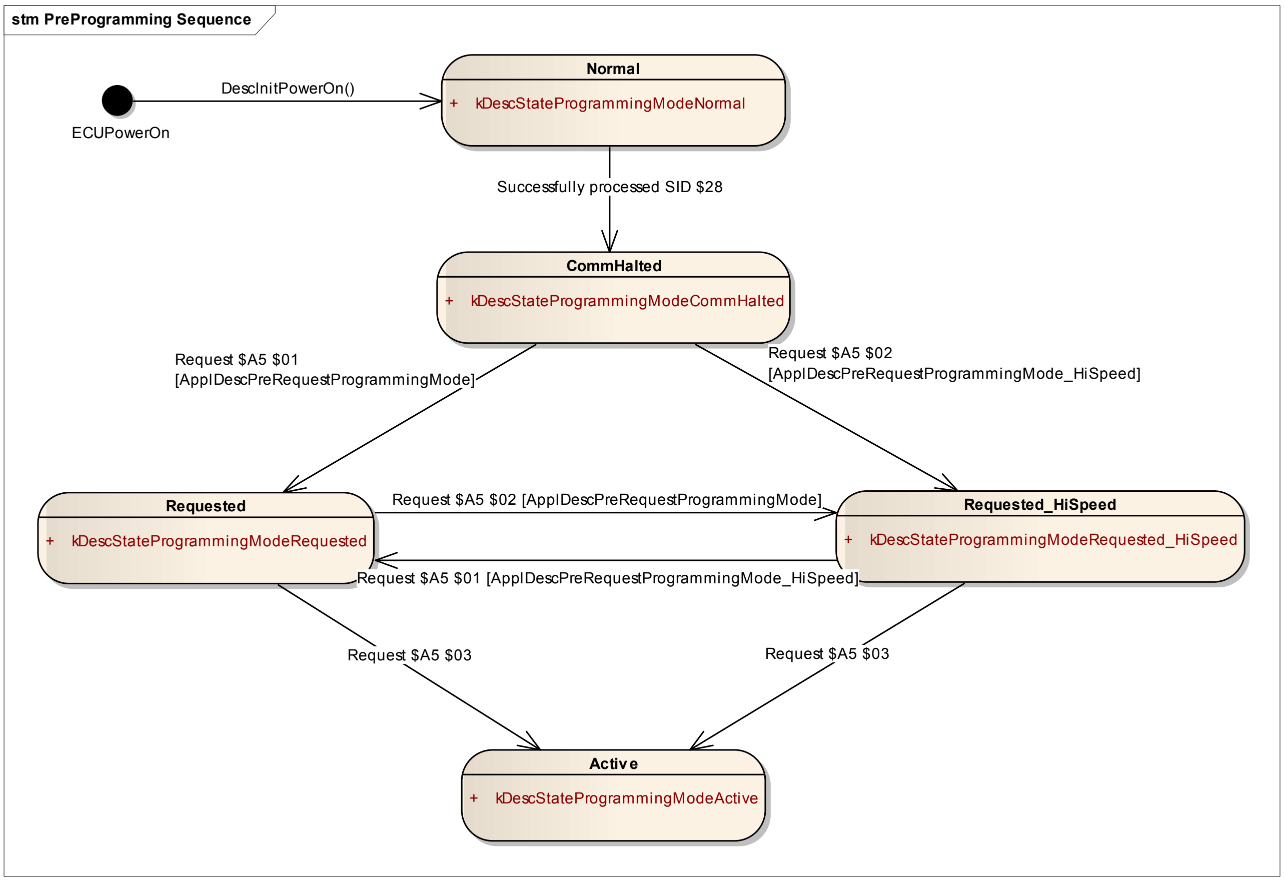

stm PreProgramming Sequence
ECUPowerOn
DescInitPowerOn()
Normal
+ kDescStateProgrammingModeNormal
Successfully processed SID $28
CommHalted
+ kDescStateProgrammingModeCommHalted
Request $A5 $01 [ApplDescPreRequestProgrammingMode]
Request $A5 $02 [ApplDescPreRequestProgrammingMode_HiSpeed]
Requested
+ kDescStateProgrammingModeRequested
Request $A5 $02 [ApplDescPreRequestProgrammingMode]
Request $A5 $01 [ApplDescPreRequestProgrammingMode_HiSpeed]
Requested_HiSpeed
+ kDescStateProgrammingModeRequested_HiSpeed
Request $A5 $03
Request $A5 $03
Active
+ kDescStateProgrammingModeActive

Figure 7-12 Programming mode flowchart

To further restrict service execution based on the current state in the programming sequence, please refer to *8 - CANdelaStudio default attribute settings* regarding the setup of the corresponding state group in CANdela studio.

7.11.1 Allowing programming mode ($A5 $01/$02)

Prior to activating programming mode, the tester must ask the ECU for permission. The application may accept or deny this request using a standard prehandler for the respective level. CANdesc will activate these prehanders per default.

Prehandler names can be changed in the CDD file or the configuration tool, for your reference the default names are:

- $A5 $01: ApplDescPreRequestProgrammingMode
  - Compatibility note: This completely replaces the dedicated callback `ApplDescMayEnterProgMode`

- $A5 $02: ApplDescPreRequestProgrammingMode_HiSpeed
  - Compatibility note: This completely replaces the dedicated callback `ApplDescMayEnterHiSpeedProgMode`

Please also refer to the general CANdesc technical reference for further information about prehandler implementation.

7.11.2 Entering programming mode ($A5 $03)

Once the entire programming mode sequence is complete, the application usually makes the jump to the bootloader (FBL) in one of two ways. Both strategies are discussed in the following sections.

7.11.2.1 FBL start on EnterProgrammingMode ($A5 $03)

If the application jumps to the FBL on this service, the ECU developer must **enable** the “Flashable ECU” option in the generation tool. CANdesc will notify the application to perform the FBL jump by calling the function *ApplDescOnEnterProgMode*. The application can use the function *DescGetHiSpeedMode* to determine if a high speed mode switch must be performed manually.

Note
If the application jumps to the FBL on this service, it must manage the bus speed switching manually.

Prototype	
`void` **`ApplDescOnEnterProgMode`** `(void)`	
Parameter	
–	-
Return code	
–	-
Functional Description	
CANdesc will call this function so the application can perform the FBL jump.	
Particularities and Limitations	
▶ To enable this notification, the ECU developer must **enable** the “Flashable ECU” option in the generation tool (the define DESC_ENABLE_FLASHABLE_ECU exists in the generated code).	
Call context	
▶ Background-loop level task context of DescStateTask().	

Table 7-9 ApplDescOnEnterProgMode

7.11.2.2 FBL start on RequestDownload ($34)

If the application jumps to the FBL on this service, the ECU developer must **disable** the option “Flashable ECU” in the generation tool. CANdesc will automatically handle the high speed switching (if necessary) to comply with the tester expectations. The application must only implement the service main-handler and perform the jump to the FBL. In order to verify that the programming mode sequence was completed successfully, the ECU developer can call the function **DescGetProgMode()** in the main-handler.

Note
The ECU developer must **disable** the option “Flashable ECU” in the generation tool when no FBL is present in the ECU.

7.11.2.3 Concluding programming mode

Once the flashing activity has concluded (through a service $20 tester request or tester present timeout), the GM/Opel diagnostic specification states that the ECU shall perform a software reset. Two scenarios are possible:

> If the ECU was actually re-flashed, the FBL should perform the reset.

> If the ECU was not actually re-flashed (the tester was re-flashing another ECU or the ECU is not flashable), CANdesc notifies the application to perform an ECU reset via the function call *ApplDescForceEcuReset*.

Prototype	
`void` **`ApplDescForceEcuReset`** `(void)`	
Parameter	
–	-
Return code	
–	-
Functional Description	
CANdesc calls this function to notify the application that it shall perform an ECU reset. It is not required that the application perform the reset within this function call since it may need to prepare first (by saving RAM variables into EEPROM, for instance).	
Particularities and Limitations	
▶ None	
Call context	
▶ Background-loop level task context of DescStateTask().	

Table 7-10 ApplDescForceEcuReset

7.11.3 Considerations when upgrading

The callback functions used by CANdesc have changed in version 6.x. The following APIs are no longer used and need to be replaced by service prehandlers:

- `ApplDescMayEnterProgMode`
- `ApplDescMayEnterHiSpeedProgMode`

7.12 Service ReadDiagnosticInformation ($A9)

This service supplies the tester with information about various DTC statuses (e.g. on change count of the DTCs, a list of all DTCs matching a given status mask, etc.).

Since the DTC storage and management is application specific, the implementation of the diagnostic service “ReadDiagnosticInformation” (RDI) ($A9) generalizes this functionality, reducing the application task to a few callback implementations.

The ECU developer should keep the following important properties in mind when designing the application:

1. When an $A9 sub-function has been requested, CANdesc cannot process another request until after the first UUDT response message has been sent.
2. While processing and sending the list of matching DTCs for an $A9 $81 request, CANdesc can process another request except $A9 $80.
3. CANdesc can process the requests $A9 $80 and $A9 $81 in parallel with $A9 $82.
4. Application callbacks are handled asynchronously.

Here is a parallel service execution matrix:

Active / Requested	$A9 $80	$A9 $81	$A9 $82	Other requests (except scheduled $AA)
$A9 $80	Not possible	Not possible	Possible after the first UUDT is sent	Not possible
$A9 $81	Not possible	Not possible	Possible after the first UUDT is sent	Not possible
$A9 $82	Not possible	Possible after the first UUDT is sent	Possible after the first UUDT is sent	Not possible
Other requests	Not possible	Possible after the first UUDT is sent	Possible after the first UUDT is sent	Not possible

Table 7-11 Service $A9 parallel execution matrix

Note
The RCR-RP responses in the sequence diagrams below are shown for better timing request-response relation understanding; however, if the application is fast enough there will be no RCR-RP response sent.

7.12.1 ReadStatusOfDTCByNumber ($A9 $80)

CANdesc processes this service as shown below.

If the application finds the requested DTC number with the given failure type byte, the following sequence will be executed:

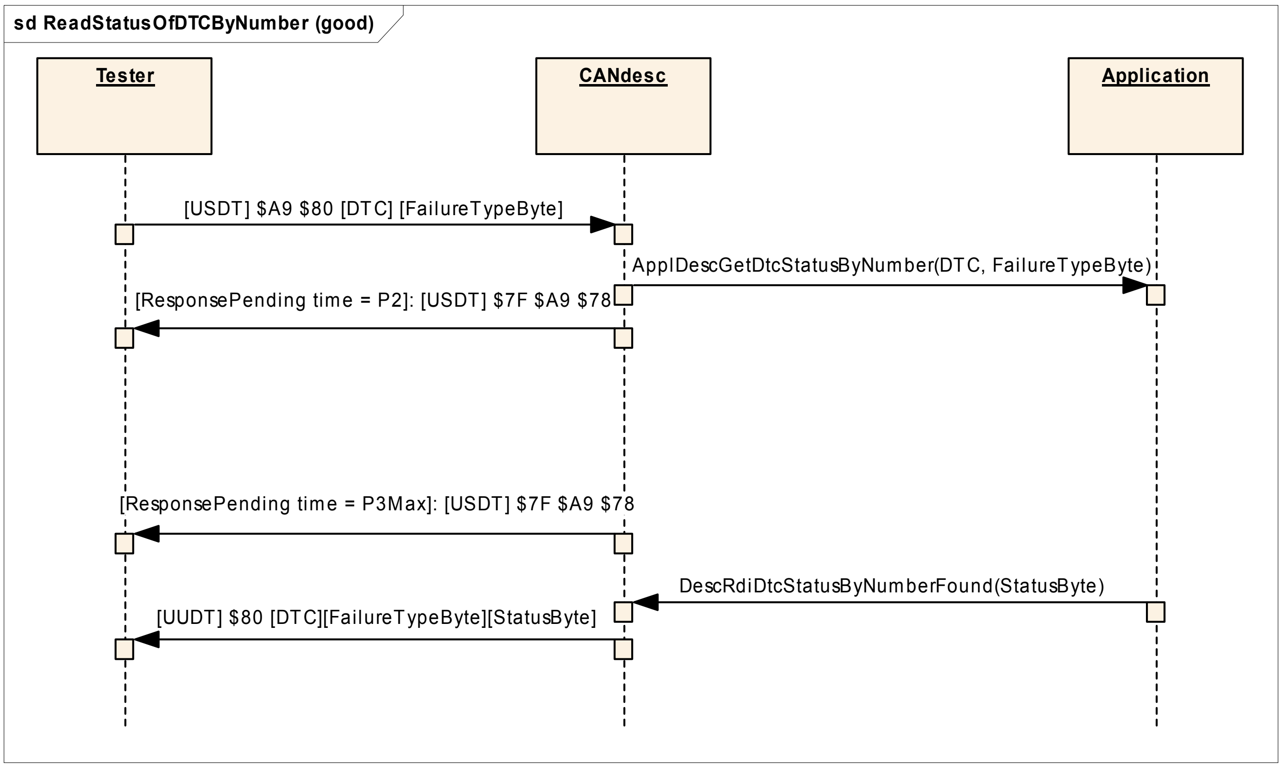

sd ReadStatusOfDTCByNumber (good)
Tester
CANdesc
Application
[USDT] $A9 $80 [DTC] [FailureTypeByte]
ApplDescGetDtcStatusByNumber(DTC, FailureTypeByte)
[ResponsePending time = P2]: [USDT] $7F $A9 $78
[ResponsePending time = P3Max]: [USDT] $7F $A9 $78
DescRdiDtcStatusByNumberFound(StatusByte)
[UUDT] $80 [DTC][FailureTypeByte][StatusByte]

Figure 7-13 Request $A9 $80 where the requested fault type combination was found

If the requested fault type combination is not supported by the ECU, the following sequence will be executed:

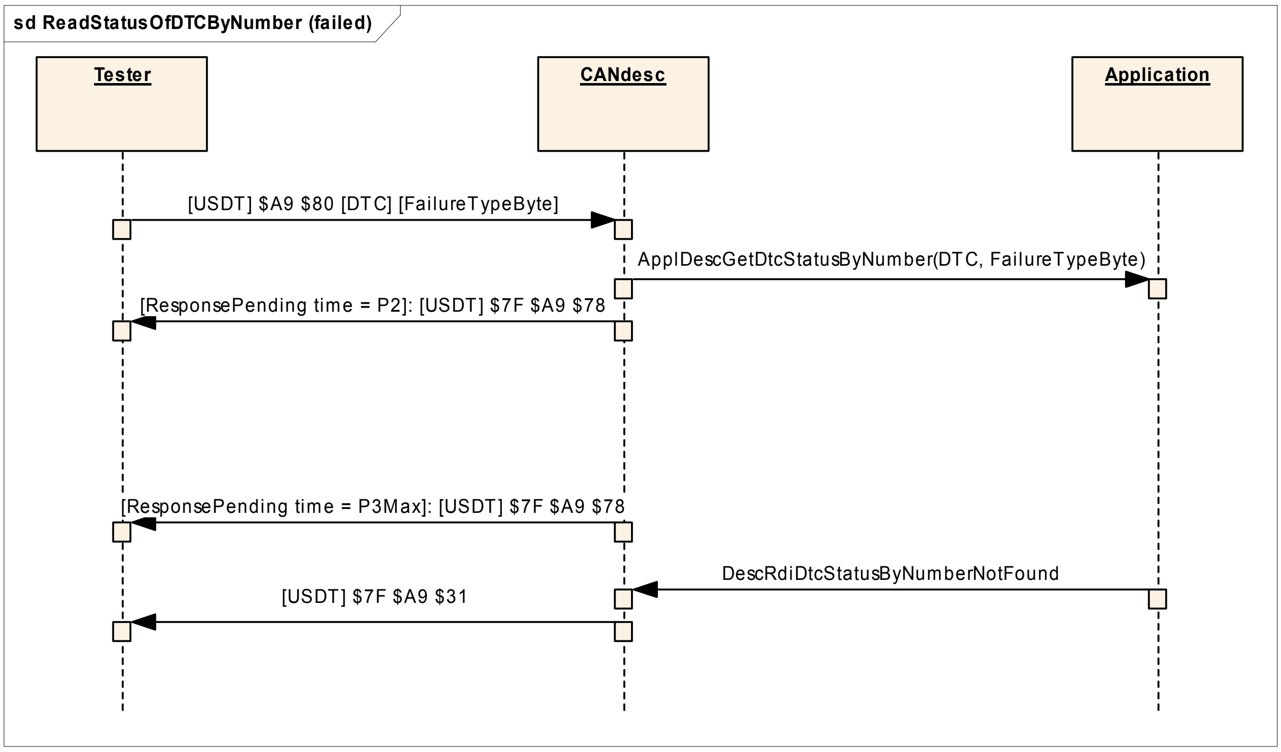

sd ReadStatusOfDTCByNumber (failed)
Tester
CANdesc
Application
[USDT] $A9 $80 [DTC] [FailureTypeByte]
ApplDescGetDtcStatusByNumber(DTC, FailureTypeByte)
[ResponsePending time = P2]: [USDT] $7F $A9 $78
[ResponsePending time = P3Max]: [USDT] $7F $A9 $78
DescRdiDtcStatusByNumberNotFound
[USDT] $7F $A9 $31

Figure 7-14 Request $A9 $80 where the requested fault type combination was not found

Here are the detailed descriptions of the APIs used in the above diagrams:

Prototype	
`void ApplDescGetDtcStatusByNumber (vuint16 dtcNum, vuint8 failureTypeByte)`	
Parameter	
`dtcNum`	The requested DTC number
`failureTypeByte`	The requested failure-type byte
Return code	
–	-
Functional Description	
CANdesc calls this function to query the application for the request combination of DTC number and failure-type byte. When the application finds this combination, it must acknowledge this by calling the function *DescRdiDtcStatusByNumberFound*. If the application cannot find a matching entry, it must acknowledge this by calling the function *DescRdiDtcStatusByNumberNotFound*.	
Particularities and Limitations	
▶ Only available if the ECU supports $A9 sub-function $80 ▶ The application can call *DescRdiDtcStatusByNumberFound* or *DescRdiDtcStatusByNumberNotFound* within this function, or it can exit this function and call one of them later on the application task level.	
Call context	
▶ Background-loop level task context (same as DescStateTask()).	

Table 7-12 ApplDescGetDtcStatusByNumber

Prototype	
`void DescRdiDtcStatusByNumberFound (vuint8 statusByte)`	
Parameter	
`statusByte`	The current status of the DTC which was passed as a parameter in the callback *ApplDescGetDtcStatusByNumber*
Return code	
–	-
Functional Description	
The application can call this function to acknowledge the successful search result of the DTC number and failure-type byte combination which were passed as parameters in the callback *ApplDescGetDtcStatusByNumber.*	
Particularities and Limitations	
▶ Only available if the ECU supports $A9 sub-function $80 ▶ CANdesc must have previously called *ApplDescGetDtcStatusByNumber*	
Call context	
▶ Background-loop level task context	

Table 7-13 DescRdiDtcStatusByNumberFound

Prototype	
`void DescRdiDtcStatusByNumberNotFound (void)`	
Parameter	
–	-
Return code	
–	-
Functional Description	
The application can call this function to acknowledge the unsuccessful search result of the DTC number and failure-type byte combination which were passed as parameters in the callback *ApplDescGetDtcStatusByNumber*	
Particularities and Limitations	
▶ Only available if the ECU supports $A9 sub-function $80 ▶ CANdesc must have previously called *ApplDescGetDtcStatusByNumber*	
Call context	
▶ Background-loop level task context	

Table 7-14 DescRdiDtcStatusByNumberNotFound

7.12.2 ReadStatusOfDTCByStatusMask ($A9 $81)

This service transmits a list of DTCs and their properties back to the tester. CANdesc uses an iteration-based process to request each element of the list from the application.

Note
The iterator parameter discussed below is only provided to help the application keep track of where it should begin/continue searching the list. The value of this parameter has no meaning to CANdesc.

CANdesc processes this service as shown below.

If the application finds DTCs that match the given status mask byte, the following sequence will be executed:

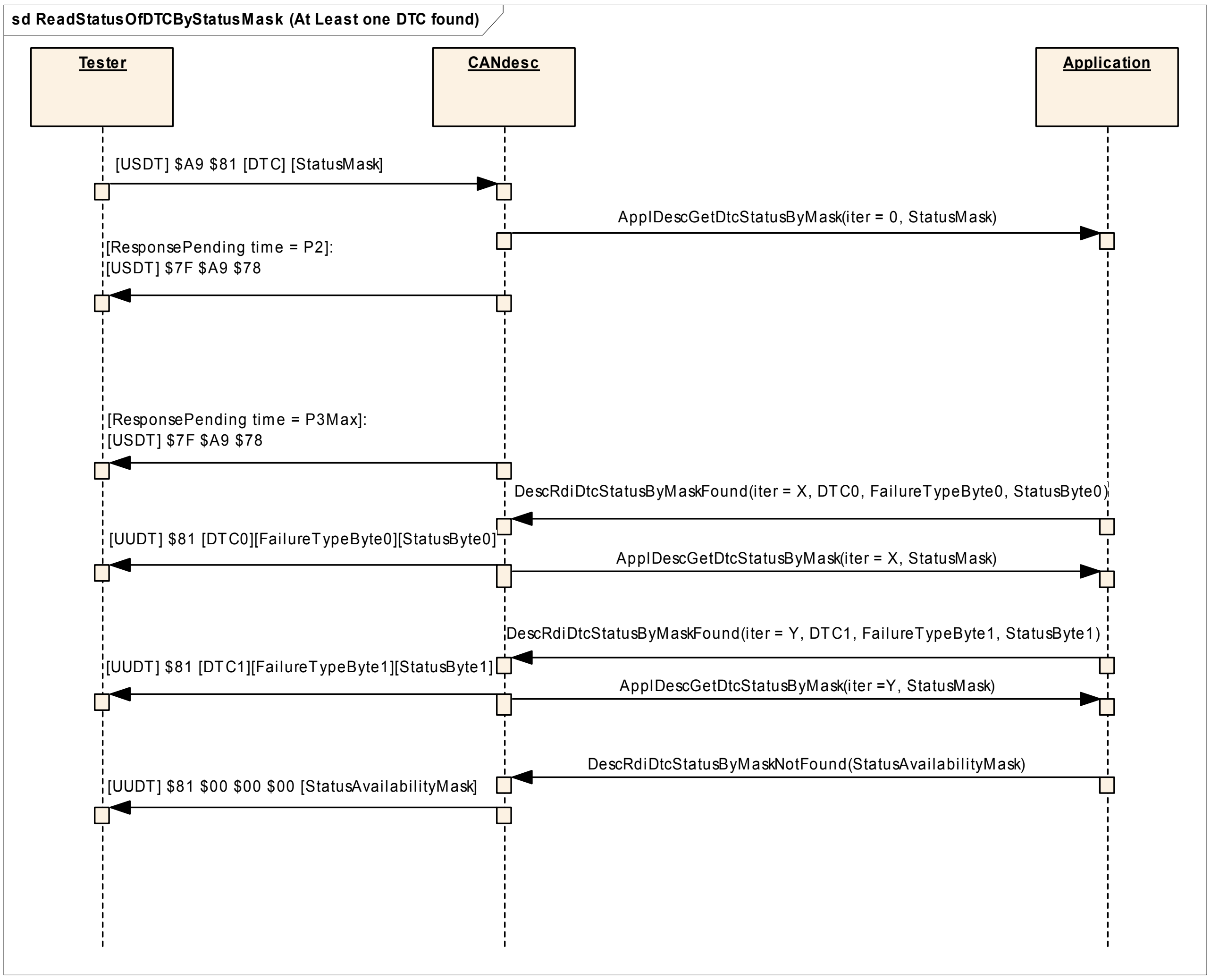

sd ReadStatusOfDTCByStatusMask (At Least one DTC found)
Tester
CANdesc
Application
[USDT] $A9 $81 [DTC] [StatusMask]
ApplDescGetDtcStatusByMask(iter = 0, StatusMask)
[ResponsePending time = P2]:
[USDT] $7F $A9 $78
[ResponsePending time = P3Max]:
[USDT] $7F $A9 $78
DescRdiDtcStatusByMaskFound(iter = X, DTC0, FailureTypeByte0, StatusByte0)
[UUDT] $81 [DTC0][FailureTypeByte0][StatusByte0]
ApplDescGetDtcStatusByMask(iter = X, StatusMask)
DescRdiDtcStatusByMaskFound(iter = Y, DTC1, FailureTypeByte1, StatusByte1)
[UUDT] $81 [DTC1][FailureTypeByte1][StatusByte1]
ApplDescGetDtcStatusByMask(iter =Y, StatusMask)
DescRdiDtcStatusByMaskNotFound(StatusAvailabilityMask)
[UUDT] $81 $00 $00 $00 [StatusAvailabilityMask]

Figure 7-15 Request $A9 $81 where the application found two DTCs with the requested status mask

Notes

1. If the application cannot send the second and the following DTCs matching the requested mask within P2ecu time, CANdesc will not send further RCR-RP messages back to the tester.
2. Although CANdesc can process another request after the first UUDT response has been sent, if that next request is either $A9 $80 or $A9 $81 the request will be rejected with negative response “ConditionsNotCorrect” ($22).
3. CANdesc can process $A9 $80 or $A9 $81 requests again once the “End of DTC report” message has been sent..

If the application cannot find any DTCs matching the requested status mask, CANdesc will only send the final UUDT response as shown below:

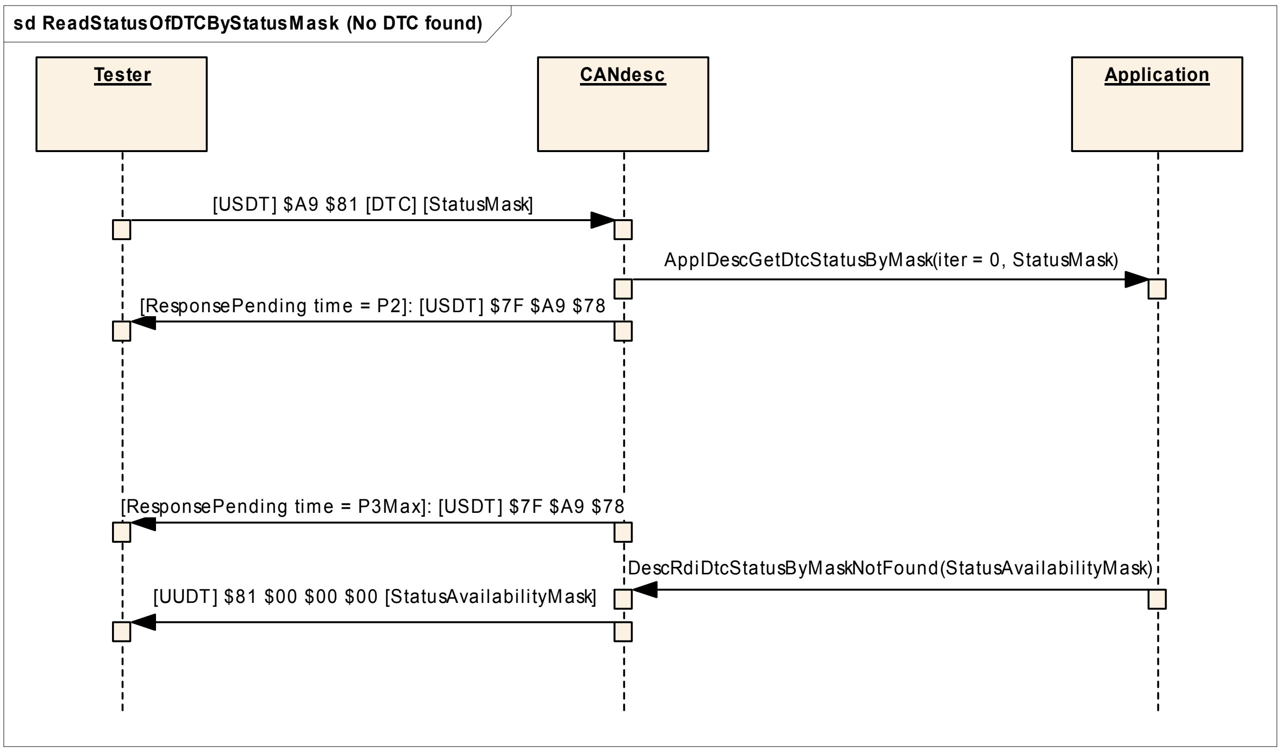

sd ReadStatusOfDTCByStatusMask (No DTC found)
Tester
CANdesc
Application
[USDT] $A9 $81 [DTC] [StatusMask]
ApplDescGetDtcStatusByMask(iter = 0, StatusMask)
[ResponsePending time = P2]: [USDT] $7F $A9 $78
[ResponsePending time = P3Max]: [USDT] $7F $A9 $78
DescRdiDtcStatusByMaskNotFound(StatusAvailabilityMask)
[UUDT] $81 $00 $00 $00 [StatusAvailabilityMask]

Figure 7-16 Request $A9 $81 where the application cannot find any DTCs with the requested status mask

Here are the detailed descriptions of the APIs used in the above diagrams:

Prototype	
`void` **`ApplDescGetDtcStatusByMask`** `(vuint16 iterPos, vuint8 statusMask)`	
Parameter	
`iterPos`	An iterator for the search start position (abstract)
`statusMask`	The status mask that the DTC shall match (OR-ed)
Return code	
–	-
Functional Description	
CANdesc calls this function to query the application for the first/next DTC number that has at least one bit in its status byte matching the parameter *statusMask*. When the application finds this combination, it must acknowledge this by calling the function *DescRdiDtcStatusByMaskFound*. If the application cannot find any (more) matching DTCs, it must acknowledge this by calling the function *DescRdiDtcStatusByMaskNotFound*. The application can use the iterator parameter as a marker to continue the search from a certain position instead of implementing a counter or state machine.	
Particularities and Limitations	
▶ Only available if the ECU supports $A9 sub-function $81 ▶ The application can call *DescRdiDtcStatusByMaskFound* or *DescRdiDtcStatusByMaskNotFound* within this function, or it can exit this function and call one of them later on the application task level.	
Call context	
▶ Background-loop level task context (same as DescStateTask()).	

Table 7-15 ApplDescGetDtcStatusByMask

Prototype	
void **DescRdiDtcStatusByMaskFound** (DescRdiDtcRecord *pDtcReport)	
Parameter	
vuint16 nextIterPos	The position of the current matching DTC
vuint16 dtcNum	The DTC number which matches the given mask
vuint8 failureTypeByte	The failure type byte of the DTC
vuint8 statusByte	The actual status byte value of the DTC
Return code	
–	-
Functional Description	
The application can call this function to acknowledge the successful search result of the statusMask which was passed as a parameter in the callback *ApplDescGetDtcStatusByMask*. The parameter pDtcReport is a pointer to a DTC property structure. Since CANdesc will immediately copy the contents of this structure, the ECU developer can use an automatic variable to save RAM.	
Particularities and Limitations	
▶ Only available if the ECU supports $A9 sub-function $81. ▶ CANdesc must have previously called *ApplDescGetDtcStatusByMask*	
Call context	
▶ Background-loop level task contex	

Table 7-16 DescRdiDtcStatusByMaskFound

Prototype	
void **DescRdiDtcStatusByMaskNotFound** (vuint8 dtcSam)	
Parameter	
dtcSam	Represents the status availability mask of the fault memory manager (i.e. which bits of the status mask are relevant for the ECU).
Return code	
–	-
Functional Description	
The application can call this function to acknowledge that there were no (more) DTCs found that match the statusMask which was passed as a parameter in the callback *ApplDescGetDtcStatusByMask*	
Particularities and Limitations	
▶ Only available if the ECU supports $A9 sub-function $81. ▶ CANdesc has previously called *ApplDescGetDtcStatusByMask.*	
Call context	
▶ Background-loop level task context	

Table 7-17 DescRdiDtcStatusByMaskNotFound

7.12.3 SendOnChangeDTCCount ($A9 $82)

This service manages the background application monitor which detects DTC count changes. In an effort to more efficiently manage this functionality, CANdesc notifies the application via function callbacks to activate/deactivate the monitor.

CANdesc processes this service as shown below.

If the tester sends a request for this service with a non-zero status mask, the DTC count monitor shall be activated:

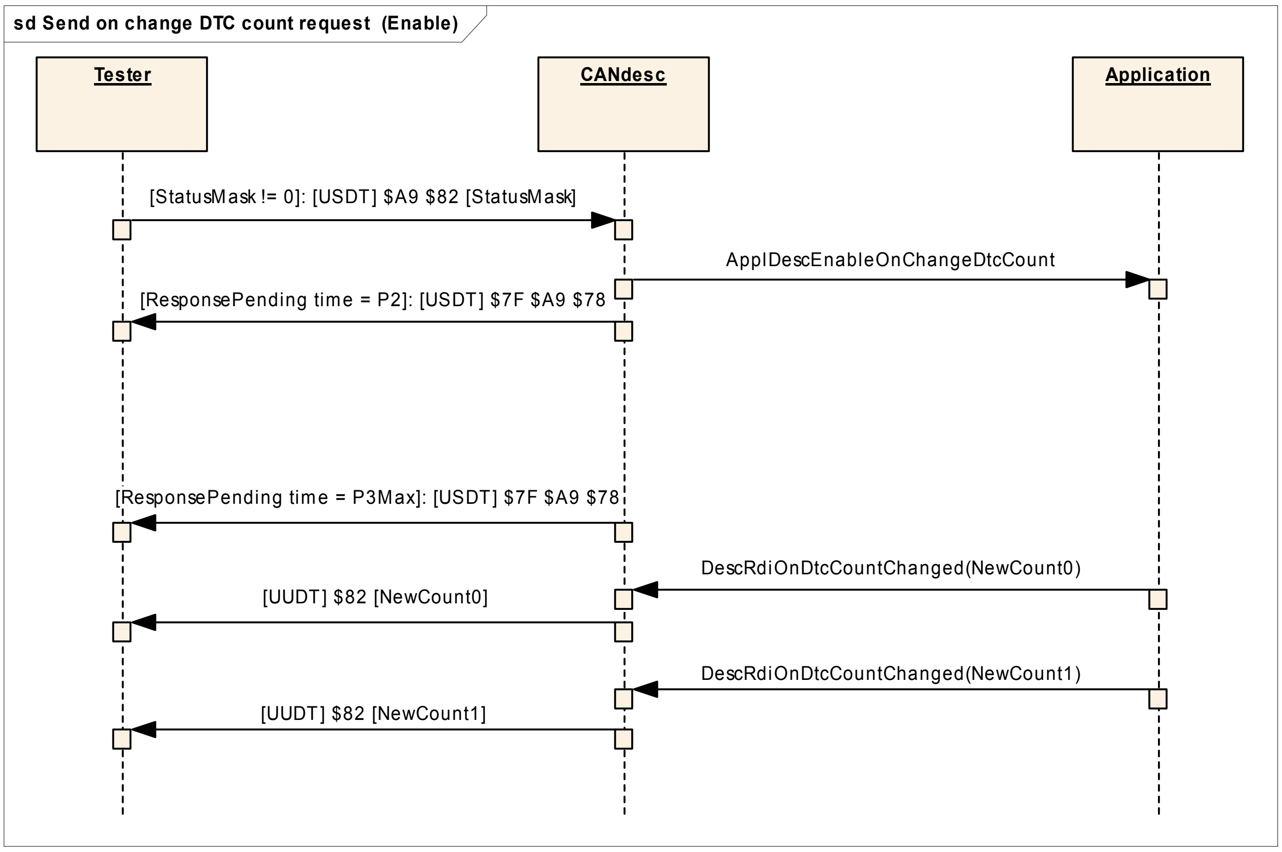

sd Send on change DTC count request (Enable)
Tester
CANdesc
Application
[StatusMask != 0]: [USDT] $A9 $82 [StatusMask]
ApplDescEnableOnChangeDtcCount
[ResponsePending time = P2]: [USDT] $7F $A9 $78
[ResponsePending time = P3Max]: [USDT] $7F $A9 $78
DescRdiOnDtcCountChanged(NewCount0)
[UUDT] $82 [NewCount0]
DescRdiOnDtcCountChanged(NewCount1)
[UUDT] $82 [NewCount1]

Figure 7-17 Request $A9 $82 with activation of the background DTC count monitor

If the tester sends a request for this service with a status mask of zero, the background monitoring shall be deactivated:

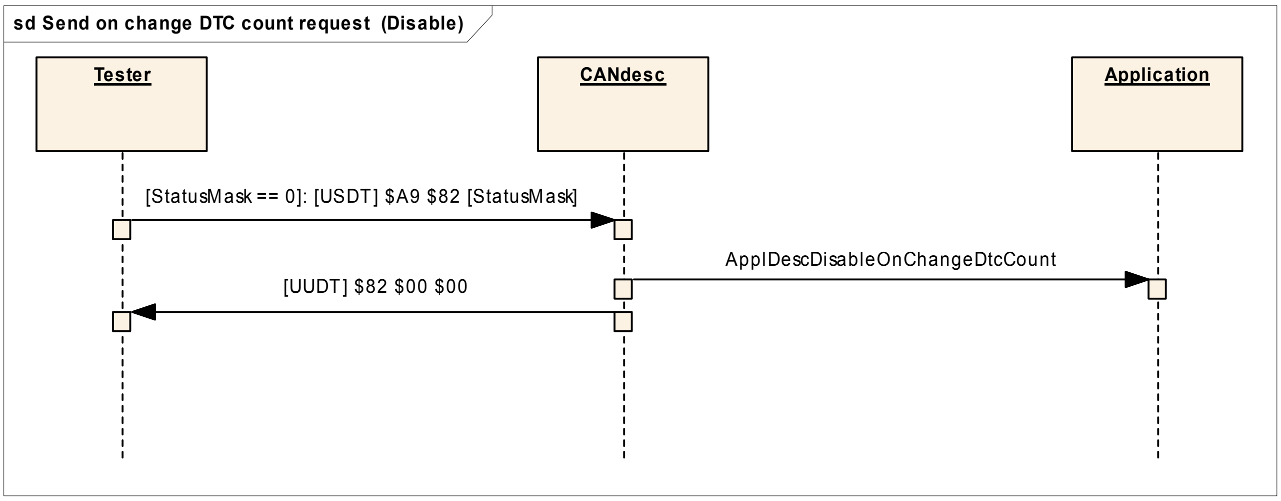

sd Send on change DTC count request (Disable)
Tester
CANdesc
Application
[StatusMask == 0]: [USDT] $A9 $82 [StatusMask]
ApplDescDisableOnChangeDtcCount
[UUDT] $82 $00 $00

Figure 7-18 Request $A9 $82 with explicit deactivation of the background DTC count monitor.

Since this service is monitored for tester present timeouts, when no more $3E requests are received the DTC count monitor shall be deactivated:

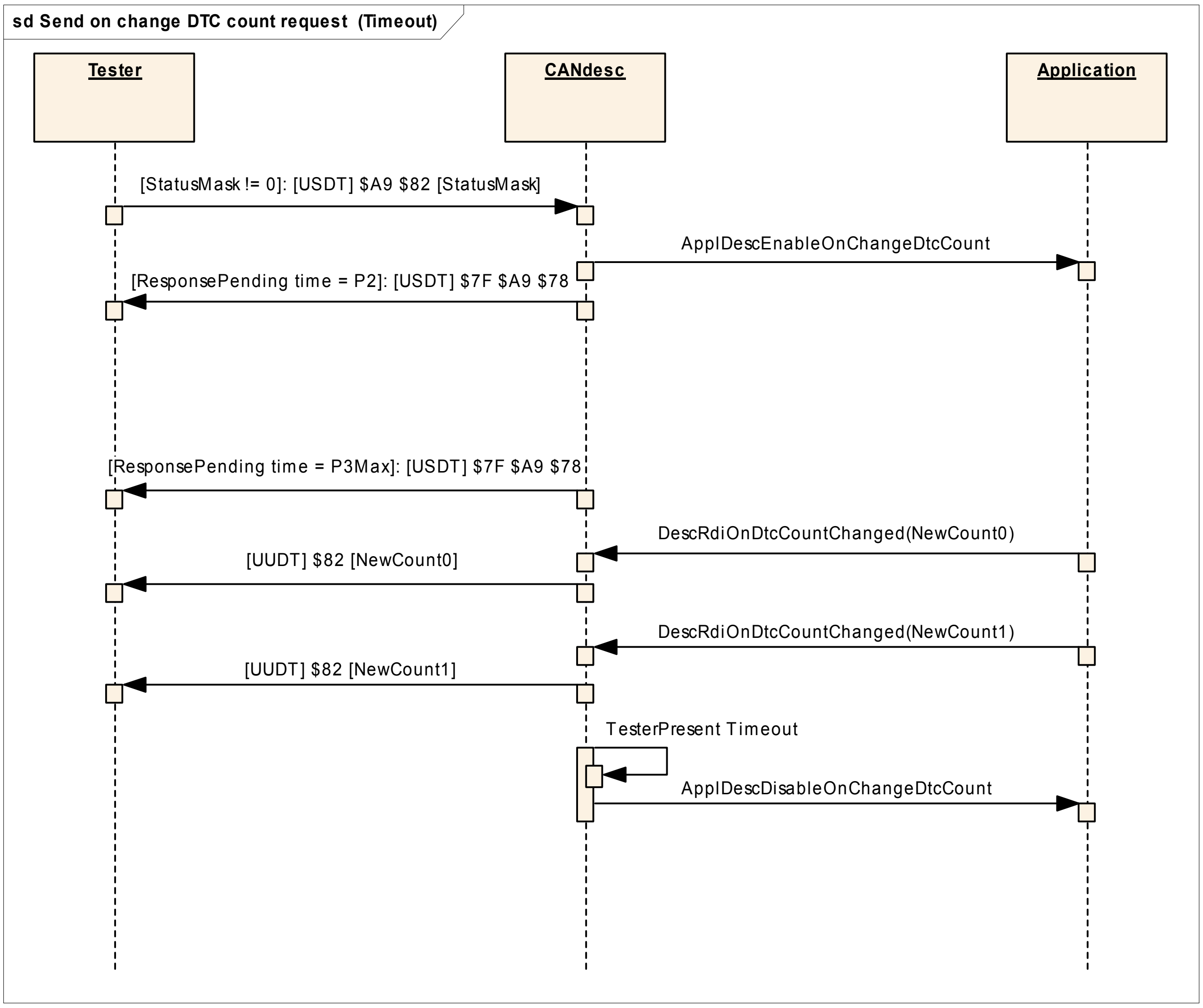

sd Send on change DTC count request (Timeout)
Tester
CANdesc
Application
[StatusMask != 0]: [USDT] $A9 $82 [StatusMask]
ApplDescEnableOnChangeDtcCount
[ResponsePending time = P2]: [USDT] $7F $A9 $78
[ResponsePending time = P3Max]: [USDT] $7F $A9 $78
DescRdiOnDtcCountChanged(NewCount0)
[UUDT] $82 [NewCount0]
DescRdiOnDtcCountChanged(NewCount1)
[UUDT] $82 [NewCount1]
TesterPresent Timeout
ApplDescDisableOnChangeDtcCount

Figure 7-19 Request $A9 $82 with deactivation of the background DTC count monitor by timeout.

Note

In addition to tester present timeout, CANdesc will react in the same way on the following events:

- The tester sends a *Service ReturnToNormalMode ($20)* request
- The application calls the function *DescRdiDeactivateOnChangeDtcCount*.

CANdesc can process this request in parallel to an $A9 $81 request after the first UUDT response has been sent.

Here are the detailed descriptions of the APIs used in the above diagrams:

Prototype	
`void` **`ApplDescEnableOnChangeDtcCount`** `(vuint8 statusMask)`	
Parameter	
`statusMask`	The status mask which the application shall apply as a filter for the DTC count monitor (i.e. which DTCs shall be monitored)
Return code	
–	-
Functional Description	
CANdesc calls this function to tell the application that it shall activate DTC count monitoring. From this point on, the application shall send the DTC count on change by calling the function *DescRdiOnDtcCountChanged*. CANdesc does not store the requested statusMask; it must be stored by the application	
Particularities and Limitations	
▶ Only available if the ECU supports $A9 sub-function $82.	
Call context	
▶ Background-loop level task context (same as DescStateTask()).	

Table 7-18 ApplDescEnableOnChangeDtcCount

Prototype	
`void` **`ApplDescDisableOnChangeDtcCount`** `(void)`	
Parameter	
–	-
Return code	
–	-
Functional Description	
CANdesc calls this function to notify the application that it shall deactivate DTC count monitoring. From this point on, the application shall not send the DTC count on change	
Particularities and Limitations	
▶ Only available if the ECU supports $A9 sub-function $82.	
Call context	
▶ Background-loop level task context (same as DescStateTask()).	

Table 7-19 ApplDescDisableOnChangeDtcCount

Prototype	
`void` **`DescRdiOnDtcCountChanged`** `(vuint16 newCount)`	
Parameter	
`newCount`	The new count of DTCs that match the mask that was passed as a parameter by the function *ApplDescEnableOnChangeDtcCount*.
Return code	
–	-
Functional Description	
The application can call this function to notify CANdesc that the DTC count has changed. CANdesc will send a UUDT response back to the tester.	
Particularities and Limitations	
▶ Only available if the ECU supports $A9 sub-function $82. ▶ CANdesc must have previously called the function *ApplDescEnableOnChangeDtcCount.*	
Call context	
▶ Background-loop level task context	

Table 7-20 DescRdiOnDtcCountChanged

Prototype	
`void` **`DescRdiDeactivateOnChangeDtcCount`** `(void)`	
Parameter	
–	-
Return code	
–	-
Functional Description	
The application can call this function to notify CANdesc that it has stopped DTC count monitoring. CANdesc will call the function *ApplDescDisableOnChangeDtcCount* to acknowledge this event.	
Particularities and Limitations	
▶ Only available if the ECU supports $A9 sub-function $82. ▶ CANdesc must have previously called the function *ApplDescEnableOnChangeDtcCount.*	
Call context	
▶ Background-loop level task context.	

Table 7-21 DescRdiDeactivateOnChangeDtcCount

7.13 Service ReadDataByPacketIdentifier ($AA)

This service allows the tester to read a DPID (data packet identifier) either once or periodically with three possible speeds (slow, medium, or fast). CANdesc completely handles this service. Only the PacketHandlers may be implemented by the application as described in section 6.7 The PacketHandler (another type of service processor).

7.13.1 Handling undefined use cases

There are some use cases which are not covered or completely described in the GM/Opel diagnostic specification. The following sections describe these cases along with the resulting CANdesc behaviour.

7.13.1.1 Service $AA handling for undefined dynamically definable DPIDs

According to GMW-3110 version 1.6, if the tester requests an undefined dynamically definable DPID with service $AA the result is undefined. In this situation, CANdesc sends a UUDT response with no actual data (byte zero is the DPID number and the remaining bytes are padded).

7.13.1.2 Service $AA handling for undefined referenced dynamically defined PIDs

As mentioned in section *7.4.1 Reading a dynamically defined PID (Parameter Identifier),* the undefined PID contained by the DPID will contain no data. In this situation, CANdesc sends a UUDT response with no actual data (byte zero is the DPID number and the remaining bytes are padded).

7.13.1.3 Service $AA handling for unaccessible referenced PIDs

A dynamically defined DPID references one or more PIDs. If any of those PIDs are not accessible to the application for any reason (e.g. security access, current ECU state, memory access error, etc.) at the time the scheduler needs the data, CANdesc sends a UUDT response with no actual data (byte zero is the DPID number and the remaining bytes are padded).

7.14 Service DeviceControl ($AE)

The application must completely implement this service. In addition, the GM/Opel diagnostic specification requires that the tester present timeout monitoring shall be activated once a DeviceControl service has been executed. Since CANdesc manages the tester present timer, the application must call the function *DescActivateS1Timer* to fulfill this requirement. The proper location for this is the PostHandler for every CPID (Control Packet Identifier) except 0x00 (reserved for “terminate device control”). To facilitate this, the ECU developer should select ‘User’ as the PostHandler attribute for these CPIDs in CANdela Studio. These PostHandlers must then evaluate the status parameter (see TechnicalReference_CANdesc) and if the result is ok (i.e. CANdesc has successfully sent the positive response) the application must call the function *DescActivateS1Timer*.

Note
To reduce ROM usage and improve run-time, the ECU developer can use the “PostHanlderOverrideName” attribute in CANdelaStudio to define a single PostHandler used by all CPIDs.

Please refer to section *8.3 Service attributes* for more details.

Here is an example how the ECU developer could implement the PostHandler:

```
void AppDescPostXXX(vuint8 iContext, vuint8 status)
{
  if (status == kDescPostHandlerStateOk)
  {
    /* Do application stuff */
    DescActivateS1Timer();
  }
}
```

Figure 7-20 PostHandler for “DeviceControl” ($AE)

Prototype	
void **DescActivateS1Timer** (void)	
Parameter	
–	-
Return code	
–	-
Functional Description	
The application can call this function to enable tester present timeout monitoring. This function is normally only used in the “DeviceControl” ($AE) service PostHandler. This function will not restart the tester present timer! Only the tester can reset the timer, by sending the “TesterPresent” ($3E) service.	
Particularities and Limitations	
▶ None	
Call context	
▶ Background-loop level task context (same as DescStateTask()).	

Table 7-22 DescActivateS1Timer

7.15 Service TesterPresent ($3E)

CANdesc implements this service by reloading the tester present timer with the original timeout value.

Caution
This service does not activate tester present timeout monitoring!

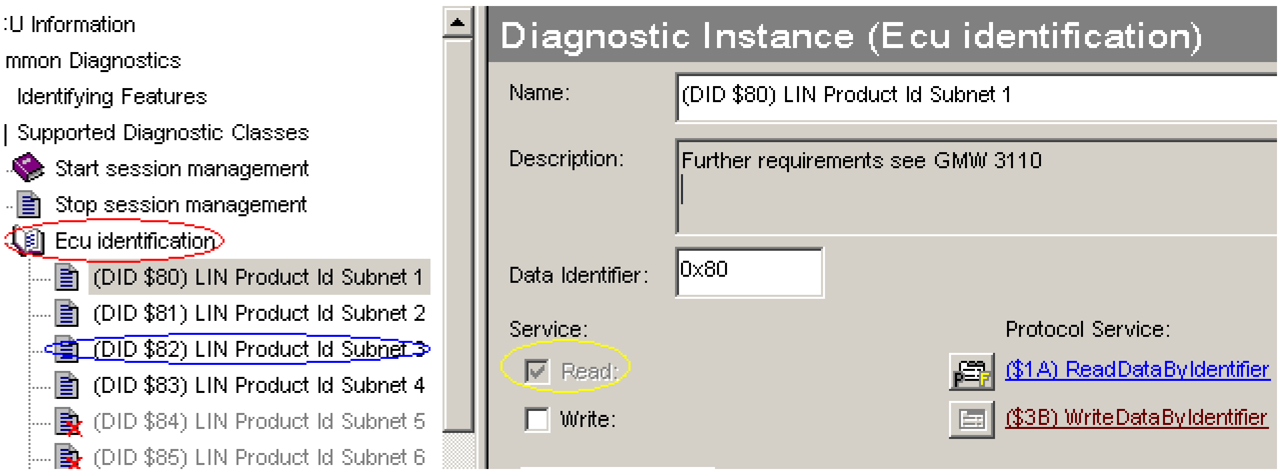
8 CANdelaStudio default attribute settings

In order to use the features of CANdesc described in this document, the ECU developer must set the CANdelaStudio attributes appropriately for the corresponding services and/or their instances as described below.

The ECU developer configures CANdelaStudio attributes on three levels:

- **Diagnostic Class** (like “Ecu identification”, “Security access”, etc.)
- **Diagnostic Instance** (like “(DID $90) Vehicle Identification Number”, “RequestProgrammingModeHighSpeed”, etc.)
- **Service** (like “Read” or “Write” on **Diagnostic Instance** “(DID $90) Vehicle Identification Number”).

The picture below illustrates these three levels in CANdelaStudio:


:U Information
mmon Diagnostics
Identifying Features
Supported Diagnostic Classes
Start session management
Stop session management
Ecu identification
(DID $80) LIN Product Id Subnet 1
(DID $81) LIN Product Id Subnet 2
(DID $82) LIN Product Id Subnet 3
(DID $83) LIN Product Id Subnet 4
(DID $84) LIN Product Id Subnet 5
(DID $85) LIN Product Id Subnet 6
Diagnostic Instance (Ecu identification)
Name:
Description:
Data Identifier:
Service:
Read:
Write:
Protocol Service:
($1A) ReadDataByIdentifier
($3B) WriteDataByIdentifier

Name: (DID $80) LIN Product Id Subnet 1

Description: Further requirements see GMW 3110

Data Identifier: 0x80

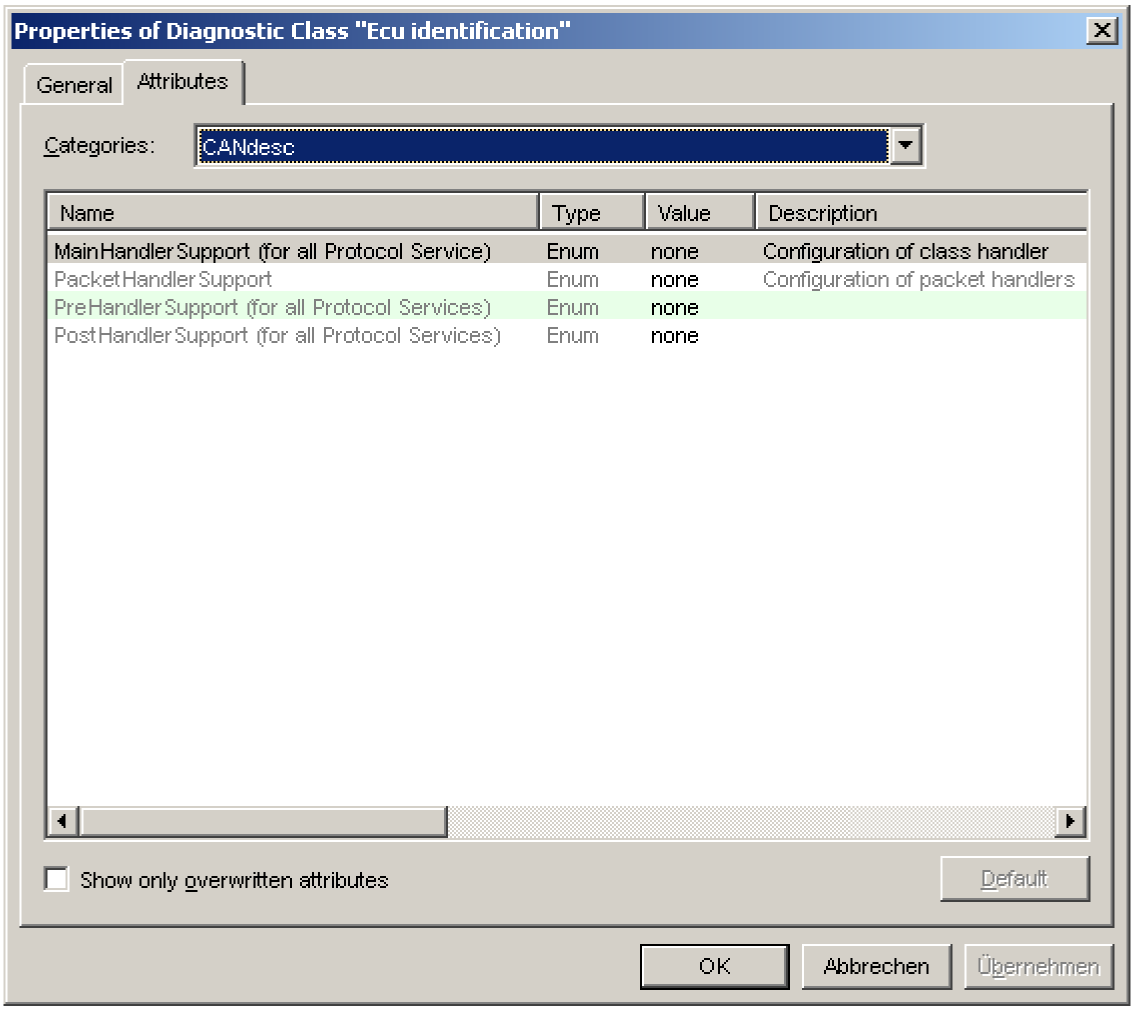
Figure 8-1 CANdelaStudio views

Legend:

RED:	Diagnostic Class
BLUE:	Diagnostic Instance
YELLOW:	Service

8.1 Diagnostic class attributes

CANdesc is designed to handle certain events on a diagnostic class basis; therefore, the ECU developer must configure the attributes on the diagnostic class level. The relevant services and their attribute configurations are listed below.


Properties of Diagnostic Class "Ecu identification"
General
Attributes
Show only overwritten attributes
Default
OK
Abbrechen
Übernehmen

Categories: CANdesc

Name	Type	Value	Description
MainHandlerSupport (for all Protocol Service)	Enum	none	Configuration of class handler
PacketHandlerSupport	Enum	none	Configuration of packet handlers
PreHandlerSupport (for all Protocol Services)	Enum	none	
PostHandlerSupport (for all Protocol Services)	Enum	none	

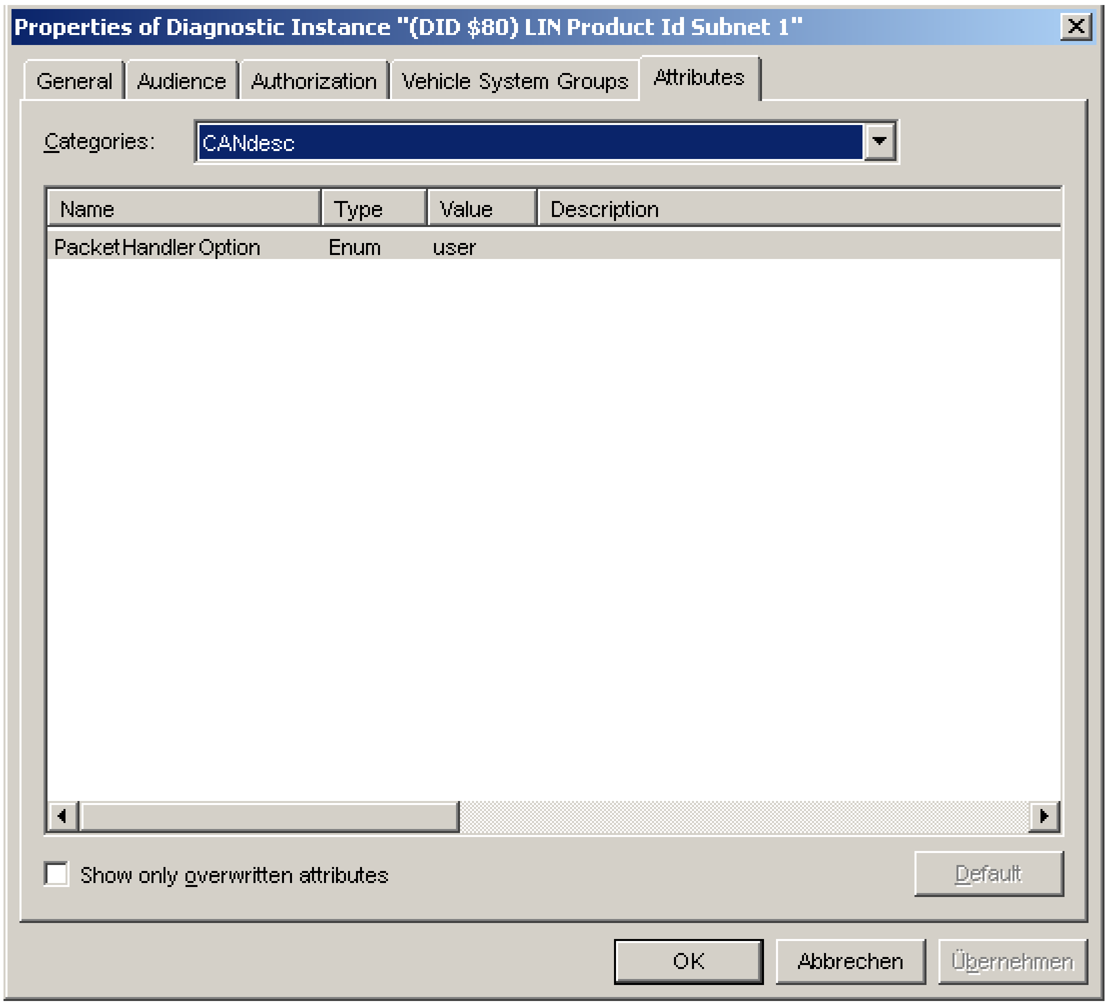
Figure 8-2 Diagnostic Class level attributes

Diagnostic Class name	MainHandler Support (for all Protocol Service)	PreHandler Support (for all Protocol Services)	PacketHandler Support	PostHandler Support (for all Protocol Services)
Failure Record Data – Parameters	**USER**	None	None	None
Dynamic Data Packets	None	None	**All**	None
Data Packet	**OEM**	None	**All**	**OEM**
Programming mode	**OEM**	None	None	**OEM**

Table 8-1 Default 'Diagnostic Class' attribute settings

8.2 Diagnostic instance attributes

CANdesc is designed to handle certain events on a diagnostic instance basis; therefore, the ECU developer must configure the attributes on the diagnostic instance level. The relevant services and their attribute configurations are listed below.



Categories: CANdesc

Name	Type	Value	Description
PacketHandler Option	Enum	user	

☐ Show only overwritten attributes

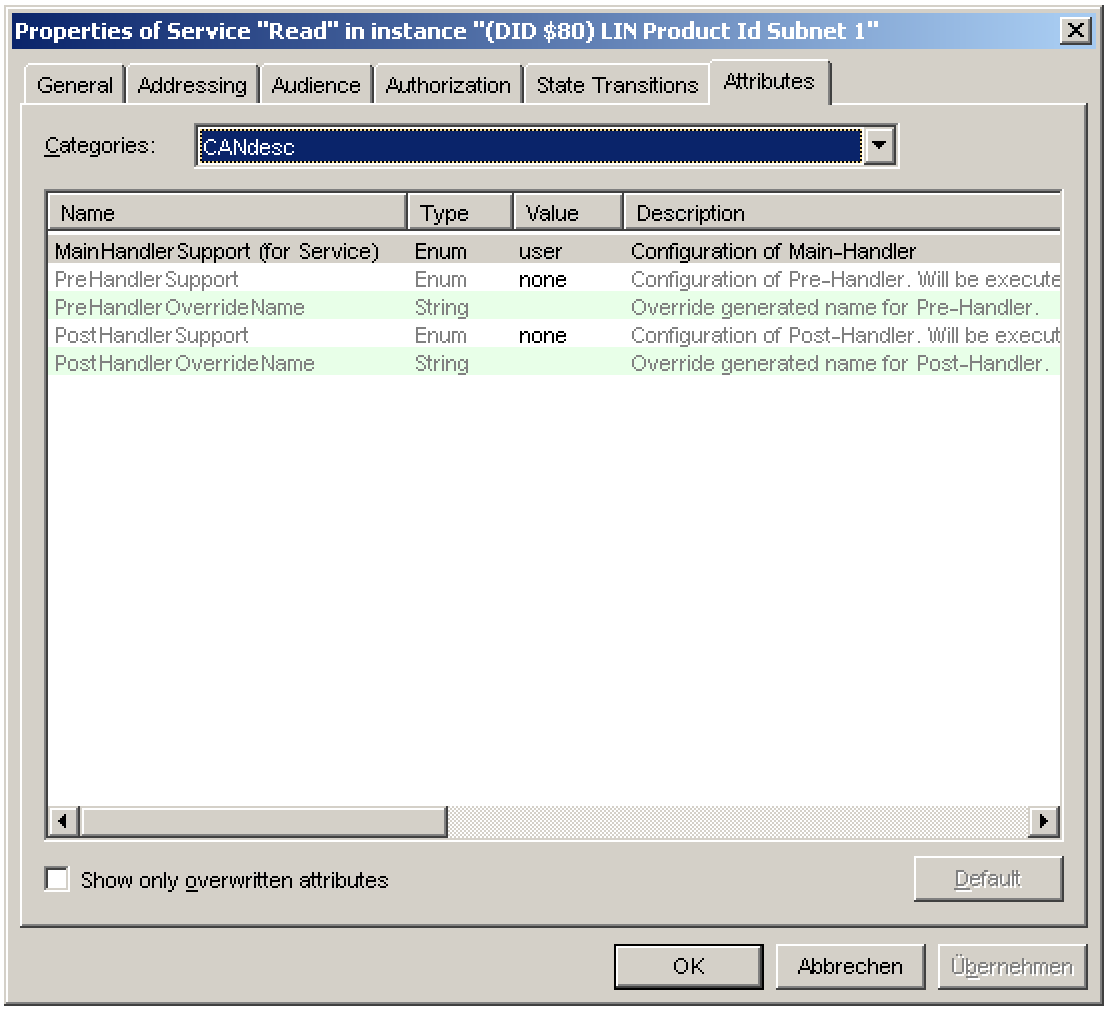
Figure 8-3 Diagnostic Instance level attributes

Diagnostic Class name where Diagnostic Instances shall be configured	PacketHandler Option	
	Default	Optional
Data Packet	**USER**	**Generated**

Table 8-2 Default ‘Diagnostic instance’ attribute settings

8.3 Service attributes

CANdesc is designed to handle certain events on a diagnostic service basis; therefore, the ECU developer must configure the attributes on the diagnostic service level. The relevant services and their attribute configurations are listed below.


Properties of Service "Read" in instance "(DID $80) LIN Product Id Subnet 1"
General
Addressing
Audience
Authorization
State Transitions
Attributes
Show only overwritten attributes
Default
OK
Abbrechen
Übernehmen

Categories: CANdesc

Name	Type	Value	Description
MainHandlerSupport (for Service)	Enum	user	Configuration of Main-Handler
PreHandlerSupport	Enum	none	Configuration of Pre-Handler. Will be execute
PreHandlerOverrideName	String		Override generated name for Pre-Handler.
PostHandlerSupport	Enum	none	Configuration of Post-Handler. Will be execut
PostHandlerOverrideName	String		Override generated name for Post-Handler.

Figure 8-4 Service related attributes

Notes:

Service (and sub-service) ID that shall be configured	MainHandler Support (for Service)	PreHandler Support	PreHandler OverrideName	PostHandler Support	PostHandler OverrideName
$10 $02	**Generated (User)**	None		**OEM**	
$10 $03	**Generated (User)**	None		**OEM**	
$20	**OEM**	None		**OEM**	
$28	**OEM** [1)]	None		None	
$3E	**Generated (User)**			**OEM**	
$A9 $80	**OEM**	**OEM**		None	
$A9 $81	**OEM**	**OEM**		None	
$A9 $82	**OEM**	None		**OEM**	
$AE $00	User	None		None	
$AE $01-$FF	User	None		**User**	One function for all sub-services (see *Service DeviceControl ($AE)*)

Table 8-3 Default ‘Service’ attribute settings

> All cells defined as “None” can optionally be replaced only by “User” attributes when desired.

> Some data access services (e.g. $22, $1A, $3B and $AA) use “Generated” MainHandler (for $AA PacketHandler) support. This eliminates the need for the application to correctly order the data in the response message (it must only return the data; CANdesc will assemble the response in the correct order)

Caution
[1)] It is not possible to implement the **MainHandler** for service **$28** in your application since CANdesc (since version 5.05.04) uses this service callback. To evaluate the service execution conditions, you can use the **PreHandler** to perform those checks and to reject the service if needed.

8.4 State group for the Programming Sequence

The CANdesc implementation for the programming mode sequence relies on a state machine that can be defined as CANdela Studio state group.

State Group Qualifier	State Qualifier	Blocked Services	Transitions
ProgrammingMode	Normal	$A5 $xx	$28 -> CommHalted
	CommHalted	$A5 $03	$A5 $01 -> Requested $A5 $02 -> Requested_HiSpeed $20 -> Normal
	Requested	None	$A5 $02 -> Requested_HiSpeed $A5 $03 -> Active $20 -> Normal
	Requested_HiSpeed	None	$A5 $01 -> Requested $A5 $03 -> Active $20 -> Normal
	Active	$A5 $xx	$20 -> Normal

Table 8-4 Programming Sequence state group

CANdesc will generate these states and transitions even if they do not exist in the CDD file. Otherwise, CANdesc will reuse what is already present, and add/remove all missing states and/or transitions.

This means you cannot change the programming sequence by means of changing the state configuration, but you **can** put the correct sequence in the CDD file (using the exact qualifiers from the table above). This opens up the possibility to block further services from executing in any of these states.

E.g. if you want to block all $AE services from executing while programming mode is 'Active', you can model this in CANdela Studio, and CANdesc will do the rest for you.

Note
Since the StateGroup 'ProgrammingMode' follows the usual state group implementation, all functionality regarding state groups is available for use. Especially the callback ApplDescOnTransitionProgrammingMode will notify your application of any state change.

9 OBD support

What is special about OBD support? GM/Opel uses extended addressing for functionally addressed enhanced diagnostic services; however, functionally addressed OBD requests must use normal addressing. Due to this, CANdesc uses two separate reception paths. Additionally, the message definition of some OBD services does not match the generic service instance generator used by CANdesc so they can only be supported on the protocol service level.

9.1 CAN identifiers

Enhanced diagnostic services

GM specifies enhanced diagnostic service requirements in GMW-3110.

CAN ID 0x101 - 11-bit extended addressing USDT

This ID is only used for functionally addressed enhanced diagnostic requests. Functionally addressed OBD requests must use CAN ID 0x7DF.

OBD services

GM uses the OBD service requirements described in ISO15031-5.

CAN ID 0x7DF - 11-bit normal addressing USDT

This ID is only used for functionally addressed OBD requests. Functionally addressed enhanced diagnostic requests must use CAN ID 0x101.

CAN ID 0x7E0-0x7EF - 11-bit normal addressing USDT

These IDs are used for physically addressed OBD requests to individual ECUs that support OBD services. ECUs using these IDs for OBD services may also elect to use the same IDs for physically addressed enhanced diagnostic services; however, ECUs that do not support OBD services may not use these IDs for enhanced diagnostic services.

9.2 Restrictions

SID $01, $02, $06, $08 and $09 must be handled at the diagnostic class level when the ‘may be combined’ property is enabled in CANdelaStudio.

9.3 CANdelaStudio default attribute settings for OBD services

9.3.1 Diagnostic classes

Powertrain diagnostic and freeze frame data (EOBD) – SID $01 / $02

Emission related trouble codes – SID $03 / $07 / $04

Test results for non-continuously monitored systems (EOBD) – SID $06

Control of on-board system, test, or component (EOBD) – SID $08

Vehicle information (EOBD) – SID $09

Service ID	PreHandler Support	MainHandler Support (On Protocol level)	PostHandler Support	PacketHandler Support
$01	None	**User**	None	None
$02	None	**User**	None	None
$06	None	**User**	None	None
$08	None	**User**	None	None
$09	None	**User**	None	None

Table 9-1 Diagnostic class specific attributes

9.4 CANgen configuration

9.4.1 DBC attribute settings for the OBD request message

A message cannot be supported by both the Interaction Layer and CANdesc. For this reason, diagnostic messages must have the attribute “GenMsgNoIalSupport” set to “yes” (or, depending on the DBC version, the attribute “GenMsgILSupport” set to “no”). The DBC author is responsible for setting this attribute.

9.4.2 CANgen version < 4.15.00

In older CANgen versions, no automatic configuration of OBD support is performed so the ECU developer must configure it manually:

For the functional OBD request message (CAN ID 0x7DF), a precopy function with the name **DescOBDReqInd** must be configured (in CANgen on the “Receive Messages” tab).

9.4.3 CANgen version ≥ 4.15.00

Since CANgen version 4.15.00, the DBC author can set the attribute ‘DiagState’ for the functional OBD request message (CAN ID 0x7DF) to ‘Yes’. In this case, CANgen will automatically configure the precopy function.

9.4.4 GENy configuration

The DBC author can set the attribute ‘DiagState’ for the functional OBD request message (CAN ID 0x7DF) to ‘Yes’. Then, CANdesc configures the precopy function automatically.

9.5 CANdesc configuration (without a Powertrain CANdela template)

If the ECU developer is not using a Powertrain template (no OBD services are available in CANdelaStudio) but the ECU must still support OBD services, a workaround is necessary to activate the OBD reception path in CANdesc. This workaround requires the ECU developer to make changes to the file “CANdelaGenAPI.ini”, which is located in the CANgen executable folder.

To override the look up result, modify:

```
[GeneratorController]
OverrideAnyObdServiceDetection = [0 - don't (default), 1 - activate OBD support]
```

The OBD services themselves will be handled by the application using the ‘user-service’ feature, which shall be enabled by the same INI file:

```
[Misc]
UseGenericUserServiceHandler = [0 - don't (default), 1 - activated]
```

Optionally, the following entry can be used for post-handler support:

```
UseGenericUserServicePostHandler = [0 - don't (default), 1 - activated]
```

10 Debug assertion codes

The GM/Opel specific implementation has optional debug code for certain situations where the ECU could “hang” or incorrect behavior could occur. Depending on the debug level chosen in the generation tool, the following cases may be checked:

Assertion name	ID(HEX)	Description
kDescAssertInvalidA9Mode	91	The module has set the $A9 sub-service iterator incorrectly, which will cause wrong buffer assignment when there is parallel processing of sub-service $82 with one of the $80 and $81 sub-functions.
kDescAssertUudtBufferAlreadyUnlocked	A0	CANdesc received confirmation that a UUDT response was successfully transmitted on the bus, but the internal state machine indicates that no transmission was in progress.
kDescAssertWrongUudtTransmitterHandle	A1	CANdesc received confirmation that a UUDT response was successfully transmitted on the bus, but the transmit handle does not match the one that CANdesc actually tried to send.
kDescAssertUudtBufferStillLocked	A2	CANdesc attempted a second UUDT transmission before the previous one was actually transmitted on the bus.
kDescAssertIllegalPostProgModeId	A3	CANdesc began the internal post processing for a programming mode request, but the requested programming mode sub-function was invalid.

Table 10-1 Debug assertion codes

11 Contact

Please visit our website for more information:

> News

> Products

> Demo software

> Support

> Training data

> Addresses

http://www.vector.com