CANdesc

Technical Reference

PSA

Version 1.00.00

Authors:	Alexander Ditte
Version:	1.00.00
Status:	released (in preparation/completed/inspected/released)

1 History

Author	Date	Version	Remarks
Alexander Ditte	2007-12-05	1.0	Initial version

2 Contents

3 Documents this one refers to…

- Technical Reference CANdesc and CANdesc (one for both)
- User Manual CANdesc

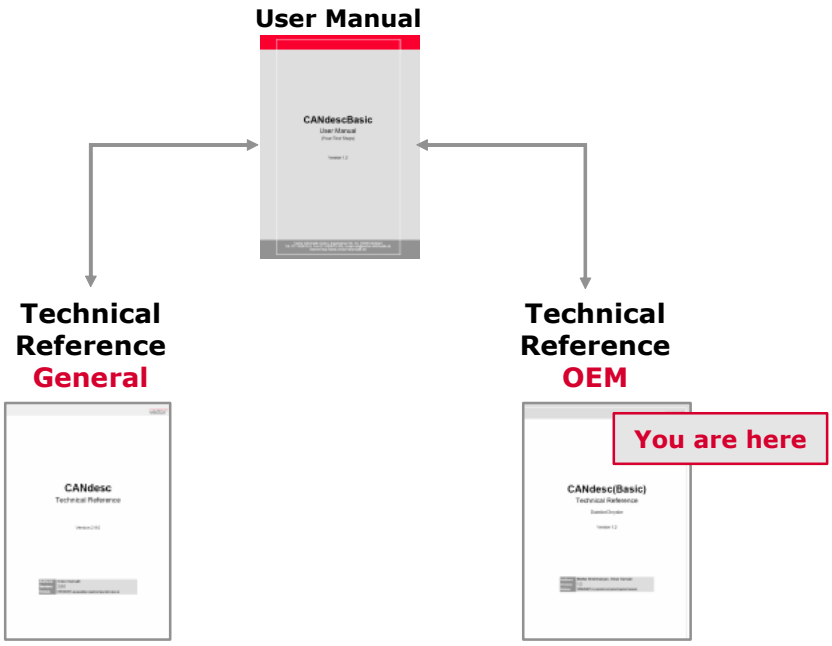

Figure 3-1 Manuals and References for CANdesc

All OEM specific topics with CANdesc are described within this technical reference and the common part (e.g. features, concepts) is in the generic technical reference.

For faster integration, refer to the user manual CANdesc.

4 CANdesc Support by Diagnostic Service

There are three possible levels of support provided by CANdesc for any specific diagnostic request – complete, assisted and basic. The level of support provided by CANdesc varies according to CANdesc functionality and user selection. All 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 includes testing support of service, but also consistency of service, sub-function and/or identifier combination. Length checking or request is performed. Validity of addressing is checked.”
Error handling is a significant part of the ECU software. All errors appearing on this level are handled inside of CANdesc.

4.1 CANdesc

Complete
Complete support means that CANdesc is capable of handling the diagnostic transaction without requesting support from (calling) the ECU application. The ECU developer need not provide any code to help implement the diagnostic feature and CANdesc handles all processing. In case a service modifies states, the application will be notified using event-callbacks – it does not have to concern about the protocol.

Assisted
Assisted support means that CANdesc is capable of fully parsing request messages and building response messages, but does not contain the logic necessary to execute the request or determine signal values. The ECU developer shall provide callbacks for CANdesc to invoke when needed to fill the logic gaps.

Basic
Basic support means that CANdesc is only capable of identifying that the ECU application shall process the request. The ECU application may have to provide logic to validate the request message (length, sub-function/parameter evaluation) and build the response byte-by-byte.

10 – Diagnostic Session Control – Assisted

The application shall provide only a function to check whether the session transition is allowed or not. All other tasks (e.g. response assembling, state management) are performed by CANdesc.

11 – Ecu Reset – Basic

The application must provide a function that resets the ECU.

14 – Clear Diagnostic Information – Basic

The application must provide a function that clears fault memory.

19 – Read DTC Information – Basic

The application must provide a function that implements the complete access to the fault memory.

22 – Read Data By Identifier – Complete/Assisted

CANdesc completely implements this service for IDs that map to global variables. Assisted support is provided for IDs that do not map to global variables (e.g. EEPROM / I/O Ports).

The so called “multiple identifier in single request” support is handled by CANdesc completely.

23 – Read Memory By Address – Basic

The application must provide a function that implements the read access to the memory.

27 – Security Access – Basic

The application must provide a function which includes the complete seed/key algorithm. The state management can be designed to be handled by CANdesc.

2E – Write Data By Identifier – Complete/Assisted

CANdesc completely implements this service for IDs that map to global variables. Assisted support is provided for IDs that do not map to global variables.

2F – Input Output Control By Identifier – Basic

The application must provide a function which implements the complete I/O control operations.

31 – Routine Control – Basic

The application must provide a function which implements the complete routine control.

34 – Request Download – Basic

The application must provide a function which implements the complete service. Instead the Vector FlashBootLoader can be used.

35 – RequestUpload – Basic

The application must provide a function which implements the complete service. Instead the Vector FlashBootLoader can be used.

36 – Transfer Data – Basic

The application must provide a function which implements the complete service. Instead the Vector FlashBootLoader can be used.

37 – Request Transfer Exit – Basic

The application must provide a function which implements the complete service. Instead the Vector FlashBootLoader can be used.

3D – Write Memory By Address – Basic

The application must provide a function that implements the write access to the memory.

3E – Tester Present – Complete

CANdesc completely implements this service.

85 – Control DTC Setting – Basic

The application must provide a function that controls the fault memory storage.

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

5 How to…

There are some important use cases you have to consider in order to fulfill the PSA diagnostic requirements. CANdesc offers special APIs and mechanisms to help your application to realize these requirements and here you will find out how to do this.

5.1 …get the Id of the current active session

PSA defines no dedicated parameter identifier for reporting the current active diagnostic session in the ECU, but if required you can use CANdesc to help you:

The CANdesc APIs for state access (i.e. DescGetStateSession()) returns the current session, but it is only an internal representation of the session not the session identifier. So you cannot use the return value of this API for that report.

To get the session Id your application shall use the API *DescGetSessionIdOfSessionState* that converts a session state internal representation into its corresponding session Id (e.g. for the default session the state is 0x01, the session Id - 0x01, but for the extended session the state representation is 0x04, the session Id – 0x03).

6 CANdesc API categories

CANdesc can be instantiated in case the configuration requires multiple diagnostic instances or/and parallel service processing. In order to offer maximum optimization for the single instance configurations there are two categories of CANdesc API.

Info
Currently Multiple Context is **NOT** relevant for PSA.

6.1 Single Context

In this configuration, the run-time and code overhead of an indexed operation mode is reduced to a minimum.

6.2 Multiple Context

In this configuration, the code is using additional handles for exact addressing and recognition of the used API.

For PSA there it depends on the configuration if a multi- or single-instance diagnostics will be used. To lower the development effort of your application and to be independent of the current configuration, please use the special API wrappers as described below.

6.3 API Context Abstraction

The following chapters describe the main use-cases where the abstraction-macros can be used.

6.3.1 Direct parameter usage

This category is usable only in API with dedicated context parameter (e.g. Pre-/PostHandlers, etc.)

6.3.1.1 DESC_CONTEXT_PARAM_VALUE

Use this macro if you have to access an array in your application that is dependent of the CANdesc running context

Example:

```
void ApplDescPreHandlerXY(DESC_CONTEXT_FORMAL_PARAM_DEF_ONLY)
{
  applDescState[DESC_CONTEXT_PARAM_VALUE] = active;
}
```

6.3.1.2 DESC_CONTEXT_PARAM_ONLY

Use this macro if you have to call a context dependent CANdesc API that have only the context handle parameter

Example: `DescRingBufferStart(DESC_CONTEXT_PARAM_ONLY);`

6.3.1.3 DESC_CONTEXT_PARAM_FIRST

Use this macro if you have to call a context dependent CANdesc API that have at least one additional to the context handle parameter

Example: `DescSetNegResponse(DESC_CONTEXT_PARAM_FIRST kDescNrcGeneralReject);`

Caution
Please note, that there shall be no comma after the macro!!!

6.3.2 Function declaration/definition usage

This category is useful if you declare/define functions that are context dependent.

6.3.2.1 DESC_CONTEXT_FORMAL_PARAM_DEF_ONLY

You can use this macro in a function declaration that is context dependent and has only parameter - the context handle.

Example: `void ApplDescPreHandlerXY(DESC_CONTEXT_FORMAL_PARAM_DEF_ONLY);`

6.3.2.2 DESC_CONTEXT_FORMAL_PARAM_DEF_FIRST

You can use this macro in a function declaration that is context dependent and has at least one additional parameter except the context handle one.

Example:

```
void ApplDescCheckSessionTransition(DESC_CONTEXT_FORMAL_PARAM_DEF_FIRST DescStateGroup newState, DescStateGroup formerState);
```

Caution
Please note, that there shall be no comma after the macro!!!

6.3.3 Universal context wrapper

This category is useful in any call-back or call of an API that is context dependent.

6.3.3.1 DESC_CONTEXT_PARAM_WRAPPER_INDEX(contextConst)

This macro is equivalent to the ***DESC_CONTEXT_PARAM_VALUE*** one but can take any expression or identifier as a parameter name.

Example: `applDescState[DESC_CONTEXT_PARAM_WRAPPER_INDEX(MyContextVariable)] = active;`

6.3.3.2 DESC_CONTEXT_PARAM_WRAPPER_ONLY(contextConst)

This macro is equivalent to the ***DESC_CONTEXT_PARAM_ONLY*** one but can take any expression or identifier as a parameter name.

Example: `DescProcessingDone(DESC_CONTEXT_PARAM_WRAPPER_ONLY(pMsgContext->iContext));`

6.3.3.3 DESC_CONTEXT_PARAM_WRAPPER_ONLY(contextConst)

This macro is equivalent to the ***DESC_CONTEXT_PARAM_FIRST*** one but can take any expression or identifier as a parameter name.

Example:

```
DescSetNegResponse(DESC_CONTEXT_PARAM_WRAPPER_FIRST(pMsgContext->iContext)
kDescNrcGeneralReject);
```

Caution
Please note, that there shall be no comma after the macro!!!

6.3.4 Utility

This category is useful in any call-back or call of an API that uses a context dependent API.

6.3.4.1 DESC_CONTEXT_PARAM_DEF_LOCAL

This defines a local context variable in a function, so you can use all the above macros without name incompatibility issues.

Example:

```
void ApplTest(void)
{
   DESC_CONTEXT_PARAM_DEF_LOCAL
   applDescState[DESC_CONTEXT_PARAM_VALUE] = active;
}
```

6.3.4.2 DESC_CONTEXT_PARAM_DUMMY_USE

If some times you don't need to use the dedicated context handle parameter, you can use this macro to avoid warnings of not referenced function parameter.

Example:

```
void ApplDescPreHandlerXY(DESC_CONTEXT_FORMAL_PARAM_DEF_ONLY)
{
  DESC_CONTEXT_PARAM_DUMMY_USE /* Ignore warnings */
}
```

7 Functions

7.1 Administrative Functions

7.1.1 DescInitPowerOn

DescInitPowerOn

Is Reentrant ☐
Is callback ☐

Prototype	
Any Context	
`void DescInitPowerOn (DescInitParam initParameter)`	
Parameter	
`initParameter`	• 'kDescPowerOnInitParam' : Default initialization
Return code	
–	-
Functional Description	
PowerOn Initialization of the CANdesc. This function has to be called before all other functions of CANdesc once after PowerOn.	
Pre-conditions	
CAN-driver via **CanInitPowerOn()** and the TransportLayer via **TpInitPowerOn()** are initialized correctly.	
Call context	
Background-loop level with global disabled interrupts	
Particularities and Limitations	
■ The **DescInitPowerOn (initParameter)** shall be called after the **TpInitPowerOn()** was called (please, refer the /TPMC/ documentation), otherwise the reserved diagnostic connection will be lost. ■ **DescInitPowerOn (initParameter)** calls internally **DescInit()** for further initializations	

7.1.2 DescInit

DescInit

Is Reentrant ☐
Is callback ☐

Prototype	
Any Context	
`void DescInit (DescInitParam initParameter)`	
Parameter	
`initParameter`	• ‘kDescPowerOnInitParam’ : Default initialization
Return code	
–	-
Functional Description	
Re-initialization of CANdesc. This function can be called to re-initialize CANdesc (e.g. after WakeUp). All internal states will be set to default.	
Pre-conditions	
CANdesc has been once initialized by DescInitPowerOn().	
Call context	
Background-loop level with global disabled interrupts	
Particularities and Limitations	
■ -	

7.2 Help functions

7.2.1 DescGetSessionIdOfSessionState

DescGetSessionIdOfSessionState

Is Reentrant ☐
Is callback ☐

Prototype	
Any Context	
`DescMsgItem` **`DescGetSessionIdOfSessionState`** `(DescStateGroup sessionState)`	
Parameter	
`sessionState`	- Shall be one of the valid session states (i.e. the value of the API *DescGetStateSession()*).
Return code	
`DescMsgItem`	- Is the corresponding session identifier value.
Functional Description	
This function provides a conversion from a session state to its corresponding session identifier (e.g. calling this function with parameter ***kDescStateSessionDefault*** will return ***0x01***).	
Pre-conditions	
-	
Call context	
-	
Particularities and Limitations	
■ -	

8 Glossary

Abbreviation	Description
CANdesc	**CAN D**iagnostic **E**mbedded **S**oftware **C**omponent Basic implementation variant
RCR-RP	**R**equest **C**orrectly **R**eceived **– R**esponse **P**ending
P2	**Maximum delay time before the first RCR-RP will be sent (50ms by UDS)**
P2* (P2Ex)	**Maximum delay time before the second RCR-RP will be sent (5000ms by UDS)**
SID	**S**ervice **Id**entifier

9 Contact

Visit our website for more information on

> News
> Products
> Demo software
> Support
> Training data
> Addresses

www.vector-informatik.com