GM Gateway Diagnostic Add-On
Technical Reference

GGDA
Version 1.8

Authors	Mishel Shishmanyan, Matthias Heil
Status	Released

Document Information

History

Author	Date	Version	Remarks
Mishel Shishmanyan	2005-11-08	1.0	Created
Mishel Shishmanyan	2006-01-31	1.1	Rework and textual corrections.
Matthias Heil	2006-02-20	1.2	Extensions for multibus support
Mishel Shishmanyan	2007-01-24	1.3	Modified: - 5.1.1 "1st step – GENtool" - 5.1.1.2 "UUDT Transmitter Configuration" - 5.1.2 "2nd step – Ggda_par.h file" Added: - 5.2 "Timings Parameter" - 5.3 "Supported Diagnostic Services" - 5.4 "Development and Integration Support" - 5.5 "Target Address Acceptance on Functional Requests"
Mishel Shishmanyan	2007-02-16	1.4	Modified: - 5.1.3.1"CAN Channel Configuration" Added: - 5.3.1 InitializeDiagnosticOperationMode ($10 $xx) - 5.3.3 ReadDiagnosticInformation ($A5 $02)
Matthias Heil	2007-05-29	1.5	Added: Infobox clarifying cdb-Attributes for functional receive message
Matthias Heil	2008-04-04	1.6	Added: Configuration aspects regarding GENy configuration Tool
Matthias Heil	2012-05-04	1.7	Added: Added description of configuration file values for mode $A9 selection
Matthias Heil	2015-05-28	1.8	Added: Precondition for DisableNormalCommunication (Sid: $28)

Reference Documents

No.	Title
[1]	TechnicalReference_CANdesc_GM_Opel.pdf

Caution

We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Figures

Tables

1 Introduction

The GGDA (GM Gateway Diagnostic Add-on) is a small diagnostic software component with reduced functionality to support only the GM's gateway non-diagnostic physical channel. This module is optimized only for this purpose and is thus very efficient. The following goals are achieved with its usage:

- implements a complete ISO-TP API for reception/transmission of diagnostic messages;
- implements an own UUDT message transmitter with timeout and data consistency protection;
- implements completely all GM required diagnostic services for a minimum diagnostic operations;
- provides CANdesc-like API for the *ReadDiagnosticInformation* service to reduce the application fault memory diagnostic API complexity;
- provides event notifications to the application for each relevant diagnostic state change (e.g. communication control disabled, wake-up link, return to normal mode, etc.);
- monitors the tester-present timeout;
- monitors the request processing time and sends automatically RCR-RP response if the final response is not registered at the P2/P2Ex times.
- implements the required interaction functionality for proper VN management;
- sends unsolicited positive response on service \$20 when the tester-present timeout event.
- has minimum ROM/RAM and run-time overhead for gateways;

2 Overview

The GGDA component takes a place parallel to the fully functional diagnostic layer CANdesc. It serves only the diagnostic services on the other communication channel on the gateway, to provide the necessary minimum of diagnostics for GM.
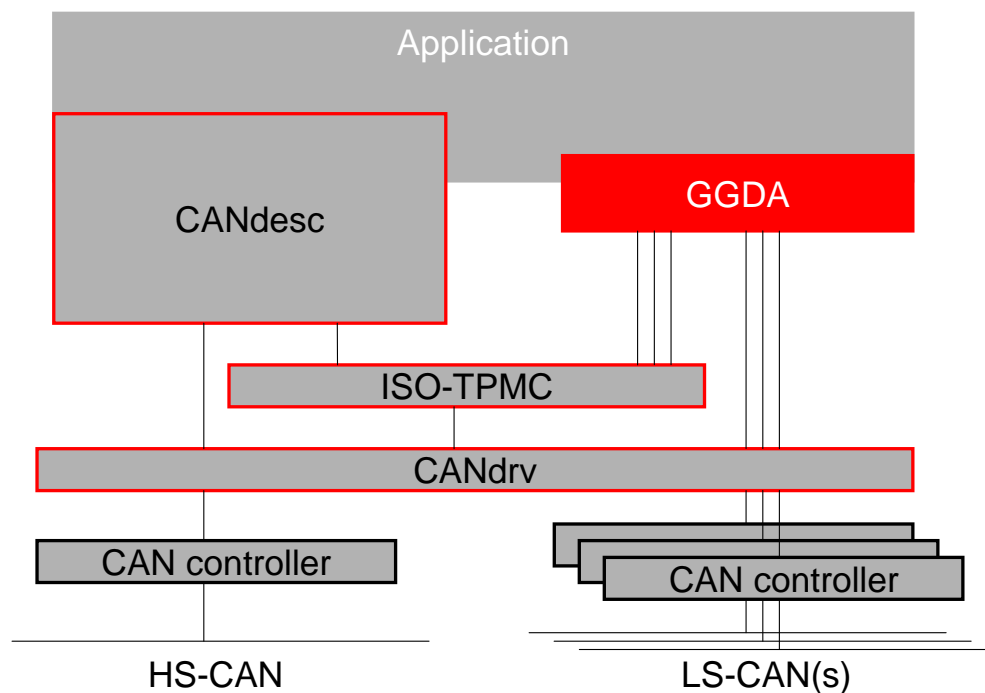
Figure 2-1 System overview.

⚠	**Caution** The GGDA component must be always placed on the low speed CANs, since the ECU will be not flash-able on this side. If the gateway is a flash-able ECU it shall be flashed only from the CANdesc side.

⚠	**Caution** GGDA is not designed to be a stand-alone component. It operates only together with the CANdesc component in the system.

3 Management Functions

To be able to work properly, the GGDA component must be managed with the following APIs as described:

3.1 Initialization

GgdaInitPowerOn

Prototype	
void **GgdaInitPowerOn** (void)	
Parameter	
-	-
Return code	
-	-
Functional Description	
In order to run properly the GGDA component must be initialized.	
Particularities and Limitations	
■ Interrupts must be disabled to guarantee consistent state machine initialization. ■ This function shall be called at ECU power-up but if needed, considering the above limitation this API may be called any time the component has to be initialized.	
Call context	
■ Any.	

3.2 Processing

The GGDA component contains two tasks for processing speed and CPU load optimization. The timer task (GgdaTimerTask) needs to be called cyclically exactly in the configured time period. The cycle time defaults to the same value as configured for CANdesc, but can be changed in ggda_par.h if needed: (example for 10ms cycle time):

```
#define kGgdaTimerMsCycle                    10   /*ms*/
```

GgdaTimerTask

Prototype	
`void GgdaTimerTask (void)`	
Parameter	
-	-
Return code	
-	-
Functional Description	
Call this function cyclically exactly in the configured time period to achieve correct timeout monitoring.	
Particularities and Limitations	
■ The function iterates through all configured channels.	
Call context	
■ Background loop with equal priority as *GgdaStateTask*().	

Table 3-1   GgdaTimerTask

GgdaStateTask

Prototype	
`void GgdaStateTask (void)`	
Parameter	
-	-
Return code	
-	-
Functional Description	
Call this function cyclically as fast as possible to achieve maximum service processing performance. Constant call time period is not necessary to be considered since there is only event polling and no time management.	
Particularities and Limitations	
■ The function iterates through all configured channels	
Call context	
■ Background loop with equal priority as *GgdaTimerTask*().	

Table 3-2   GgdaStateTask

4 What is inside

The GGDA software component implements completely all GM required services on the non-diagnostic side of the gateway. Some events and call-backs are necessary to be implemented by the application, since they reference other ECU application's specific functionality. Below you will find the implementation description of the GGDA referenced by the diagnostic service specification.

4.1 InitializeDiagnosticOperationMode (Sid $10)

This service is only needed if the ECU must support one of the following sub-functions:

4.1.1 DisableAllDTCs ($02)

In order to activate this feature, please see the configuration details in chapter 5.1.3.

If this feature is activated and the tester sends a valid request of this service, the following call-back function will be called:

ApplGgdaOnDisableAllDtc

Prototype	
void **ApplGgdaOnDisableAllDtc** (void)	
Parameter	
-	-
Return code	
-	-
Functional Description	
Notifies about requested service to disable the DTC setting.	
Particularities and Limitations	
■ Available only if configured.	
Call context	
■ Called from the **GgdaStateTask** context.	

Table 4-1   ApplGgdaOnDisableAllDtc

4.1.2   WakeUp Link ($04)

In order to activate this feature, please see the configuration details in chapter 5.1.3.

If this feature is activated and the tester sends a valid request of this service, the following call-back function will be called:

ApplGgdaOnWakeUpLink

Prototype	
void **ApplGgdaOnWakeUpLink** (void)	
Parameter	
-	-
Return code	
-	-
Functional Description	
Notifies about requested service for wake up.	
Particularities and Limitations	
■ Available only if configured.	
Call context	
■ Called from the **GgdaStateTask** context.	

Table 4-2   ApplGgdaOnWakeUpLink

4.2   ReadEcuIdentification (Sid $1A)

From this service only one identifier is necessary to be supported ($B0) and it is completely implemented in GGDA.

Since this service returns the ECU address, you have to configure the correct ECU address id. The configuration defaults to the address configured for CANdesc.

In CANgen configurations the value can be changed in Ggda_par.h (example: ECU address is 0x3B):

```
#define kGgdaEcuNumber                          0x3B
```

In GENy configurations this value is configurable in the GENy tool.

4.3 ReturnToNormalMode (Sid: $20)

If a valid request of this service has been received, or a timeout of the tester-present timer has been detected, the following APIs will be called:

ApplGgdaOnReturnToNormalMode

Prototype	
void **ApplGgdaOnReturnToNormalMode** (GgdaContextIndexType context)	
Parameter	
- context	- Identifies the channel on which the GGDA has restored normal mode. You can safely ignore this parameter if the GGDA only handles one channel.
Return code	
-	-
Functional Description	
Notifies about leaving all activities activated from the tester (e.g. CommControlHalted, WakeUpLink, etc.) up to now. Within this function call you have to enable the DTC setting (if previously disabled).	
Particularities and Limitations	
■ None	
Call context	
■ Called from the **GgdaStateTask** context.	

Table 4-3   ApplGgdaOnReturnToNormalMode

If the ECU was simulating the flash-process, the following function will be called to perform an ECU reset (within or outside the callback):

ApplGgdaForceEcuReset

Prototype	
void **ApplGgdaForceEcuReset** (void)	
Parameter	
-	-
Return code	
-	-
Functional Description	
Notifies about ECU reset execution. The reset can be done immediately or later (if some EEPROM writing is needed). If a diagnostic response was necessary, it already has been sent before this function is called.	
Particularities and Limitations	
■ None	
Call context	
■ Called from the **GgdaStateTask** context.	

Table 4-4   ApplGgdaForceEcuReset

4.4 DisableNormalCommunication (Sid: $28)

Once a valid service of this type has been requested the GGDA will (optionally) request application permission to process it:

ApplGgdaMayDisableNormalComm

Prototype	
vuint8 **ApplGgdaMayDisableNormalComm** (GgdaContextIndexType context)	
Parameter	
- context	- Identifies the channel on which the request has been received. You can safely ignore this parameter if the GGDA only handles one channel.
Return code	
vuint8	*kDescOk* – if the communication control request can be accepted or *kDescFailed* – if the communication control request shall **NOT** be accepted.
Functional Description	
Once a valid diagnostic request $28 was received this function will be called. The application shall decide whether to accept the communication control request, or to reject it. Rejection will always result in NRC 'ConditionsNotCorrect'	
Particularities and Limitations	
■ Entering NormalCommDisabled can fail even if the application has allowed the service. Use ApplGgdaOnDisableNormalComm as notification of the state change. Also, your code may not depend on ApplGgdaOnDisableNormalComm being called after this function has returned kDescOk. ■ This callback is called only when enabled in the configuration tool.	
Call context	
■ Called from the **GgdaStateTask** context.	

Table 4-5   ApplGgdaMayDisableNormalComm

Once the service has been processed, the application will be notified about the new state with the call of:

Prototype	
void **ApplGgdaOnDisableNormalComm** (GgdaContextIndexType context)	
Parameter	
- context	- Identifies the channel on which the GGDA disabled communication. You can safely ignore this parameter if the GGDA only handles one channel.
Return code	
-	-
Functional Description	
Notifies about entering in disabled normal communication mode. The GGDA has already disabled communication when this function is called.	
Particularities and Limitations	
■ None	
Call context	
■ Called from the **GgdaStateTask** context.	

Table 4-6   ApplGgdaOnDisableNormalComm

4.5   TesterPresent (Sid $3E)

This service is completely handled by GGDA. The valid request resets the tester-present timeout timer.

	FAQ
	Functionally requested valid $3E resets always the tester present timeout timer (even if currently another service processing is in progress). The timeouts are supervised independently for each configured channel**.**

4.6 ProgrammingMode (Sid: $A5)

The GGDA implements the whole GM flash process preparation flow, considering the correct services' requests order. The application will be asked only if it is ready to accept the entering in program mode request. Once accepted, the GGDA component handles the remaining services.

> **Info**
> The flowcharts for supporting this service are equal to the one used in CANdesc- just the API names are different. Please, refer to [1] for details.

> **Info**
> Please, note that the same return type **vuint8** is used here to allow you to implement a central condition check API to your application.

4.6.1 RequestProgrammingMode ($01)

ApplGgdaMayEnterProgMode

Prototype	
`vuint8` **`ApplGgdaMayEnterProgMode`** `( GgdaContextIndexType context )`	
Parameter	
- context	- Identifies the channel on which the request has been received. You can safely ignore this parameter if the GGDA only handles one channel.
Return code	
vuint8	*kDescOk* – if the programming mode shall be accepted or *kDescFailed* –if the programming mode shall **NOT** be accepted.
Functional Description	
Once the diagnostic request $A5 $01 was received from the CANdesc module this function will be called. The application shall decide whether to accept the requested programming mode or to reject it.	
Particularities and Limitations	
■ None	
Call context	
■ Called from the **GgdaStateTask** context.	

Table 4-7   ApplGgdaMayEnterProgMode

4.6.2 RequestHiSpeedProgrammingMode ($02)

In order to activate this feature, please see the configuration details in chapter 5.1.3.

ApplGgdaMayEnterHiSpeedProgMode

Prototype	
`vuint8` **`ApplGgdaMayEnterHiSpeedProgMode`** `( GgdaContextIndexType context )`	
Parameter	
- context	- Identifies the channel on which the request has been received. You can safely ignore this parameter if the GGDA only handles one channel.
Return code	
vuint8	*kDescOk* – if the programming mode shall be accepted or *kDescFailed* –if the programming mode shall **NOT** be accepted.
Functional Description	
Once the diagnostic request $A5 $02 was received from the CANdesc module this function will be called. The application shall decide whether to accept the requested programming mode or to reject it.	
Particularities and Limitations	
■ None	
Call context	
■ Called from the **GgdaStateTask** context.	

Table 4-8   ApplGgdaMayEnterHiSpeedProgMode

4.6.3 EnableProgrammingMode ($03)

Once the requested programming mode has been accepted, the GGDA module performs just a state transition here to allow the ECU programming mode compliant behaviour (e.g. no response on $A5 $03, no response on the next request $20 and suppressed unsolicited positive response on Sid $20 on tester-present timeout.

4.7   ReadDiagnosticInformation (Sid: $A9)

The GGDA implements the whole GM fault memory reading process flow. The application will be asked with the usage of signal API to provide the necessary information and if iteration is needed, the API will be used multiple times.

> **Info**
> The flow-charts for supporting this service are equal to the one used in CANdesc – just the API names are different. Please, refer to [1] for details.

4.7.1   ReadStatusOfDTCByDTCNumber ($80)

In order to activate this feature, please see the configuration details in chapter 5.1.3.

ApplGgdaGetDtcStatusByNumber

Prototype	

`void` **`ApplGgdaGetDtcStatusByNumber`** `( GgdaContextIndexType context`
`                              vuint16 dtcNum,`
`                              vuint8 failureTypeByte )`	

Parameter	
- context - dtcNum - failureTypeByte	- Identifies the channel on which the request has been received. You can safely ignore this parameter if the GGDA only handles one channel. - The DTC which status will be checked from the application - The failure type combination.
Return code	
-	-
Functional Description	
Once the diagnostic request $A9 $80 was received from the CANdesc module this function will be called.	
Particularities and Limitations	
■ Available only if mode $A9 is enabled and level $80 is supported	
Call context	
■ Called from the **GgdaStateTask** context.	

Table 4-9   ApplGgdaGetDtcStatusByNumber

GgdaRdiDtcStatusByNumberFound

Prototype	
`void `**`GgdaRdiDtcStatusByNumberFound`**` ( GgdaContextIndexType context,` `                               vuint8 statusByte )`	
Parameter	
- context - statusByte	- Identifies the channel for which the request is answered. Please pass '0' (zero) here if the GGDA only handles one channel. - The found DTC's status byte value.
Return code	
-	-
Functional Description	
Once the application has been requested to find a specific DTC through the call of *ApplGgdaGetDtcStatusByNumber* function, it shall confirm the status of the search process with this API if such a DTC has been found.	
Particularities and Limitations	
■ Available only if mode $A9 is enabled and level $80 is supported	
Call context	
■ Any	

Table 4-10   GgdaRdiDtcStatusByNumberFound

GgdaRdiDtcStatusByNumberNotFound

Prototype	
`void `**`GgdaRdiDtcStatusByNumberNotFound`**` ( GgdaContextIndexType context )`	
Parameter	
- context	- Identifies the channel for which the request is answered. Please pass '0' (zero) here if the GGDA only handles one channel.
Return code	
-	-
Functional Description	
Once the application has been requested to find a specific DTC through the call of *ApplGgdaGetDtcStatusByNumber* function, it shall confirm the status of the search process with this API if **NO** such a DTC has been found.	
Particularities and Limitations	
■ Available only if mode $A9 is enabled and level $80 is supported	
Call context	
■ Any	

Table 4-11   GgdaRdiDtcStatusByNumberNotFound

4.7.2 ReadStatusOfDTCByStatusMask ($81)

i	**Info** Please, not that the same data type **DescRdiDtcRecord** is used here to allow you to implement a central data acquisition API to your application.

ApplGgdaGetDtcStatusByMask

Prototype

```void ApplGgdaGetDtcStatusByMask (GgdaContextIndexType context,```
 ```   vuint16 iterPos,```
 ```   vuint8 statusMask)```

Parameter	
- context - iterPos - statusMask	- Identifies the channel for which the request is answered. You can safely ignore this parameter if the GGDA only handles one channel. - The next DTC status scanner start position (assumed normal iterator); - The searched status mask.
Return code	
-	-
Functional Description	
Once the diagnostic request $A9 $81 was received from the CANdesc module this function will be called. It will be called as long as the application responds each time with *GgdaRdiDtcStatusByMaskFound* until the application some time responses with *GgdaRdiDtcStatusByMaskNotFound*.	
Particularities and Limitations	
■ The application is responsible to hold the DTC information separate for each channel. Requests on different channels may request different status masks.	
Call context	
■ Called from the **GgdaStateTask** context.	

Table 4-12   ApplGgdaGetDtcStatusByMask

GgdaRdiDtcStatusByMaskFound

Prototype	

`void `**`GgdaRdiDtcStatusByMaskFound`**` (GgdaContextIndexType context,`
                    `const DescRdiDtcRecord * const`
 `pDtcReport)`	

Parameter	
- context - pDtcReport	- Identifies the channel for which the request is answered. Please pass '0' (zero) here if the GGDA only handles one channel. - The found DTC's response relevant information.
Return code	
-	-
Functional Description	
Once the application has been requested to find a specific DTC through the call of *ApplGgdaGetDtcStatusByMask* function, it shall confirm the status of the search process with this API if such a DTC has been found. Additionally all required by the `DescRdiDtcRecord` data structure information shall be initialized as described in [1].	
Particularities and Limitations	
■ Available only if mode $A9 is enabled.	
Call context	
■ Any	

Table 4-13 GgdaRdiDtcStatusByMaskFound

GgdaRdiDtcStatusByMaskNotFound

Prototype	
`void `**`GgdaRdiDtcStatusByMaskNotFound`**` (GgdaContextIndexType context` `vuint8 dtcSam)`	
Parameter	
- context - dtcSam	- Identifies the channel for which the request is answered. Please pass '0' (zero) here if the GGDA only handles one channel. - The DTC Status Availability Mask.
Return code	
-	-
Functional Description	
Once the application has been requested to find a specific DTC through the call of *ApplGgdaGetDtcStatusByMask* function, it shall confirm the status of the search process with this API if no (more) such a DTC has been found. Additionally the status availability mask shall be as described in [1].	
Particularities and Limitations	
■ Available only if mode $A9 is enabled	
Call context	
■ Any	

Table 4-14   GgdaRdiDtcStatusByMaskNotFound

4.7.3 SendOnChangeDTCCount ($82)

ApplGgdaEnableOnChangeDtcCount

Prototype	
void **ApplGgdaEnableOnChangeDtcCount** (GgdaContextIndexType context vuint8 statusMask)	
Parameter	
- context - statusMask	- Identifies the channel for which the request is answered. You can safely ignore this parameter if the GGDA only handles one channel. - the monitored DTCs' matching status mask.
Return code	
-	-
Functional Description	
Once the diagnostic request $A9 $82 was received from the CANdesc module this function will be called. The application shall store the statusMask parameter value and start its background DTC count evaluation algorithm.	
Particularities and Limitations	
■ The application is responsible to support several concurrent counting algorithms. Requests on different channels may request different status mask ■ Available only if mode $A9 is enabled and level $82 is supported	
Call context	
■ Called from the **GgdaStateTask** context.	

Table 4-15   ApplGgdaEnableOnChangeDtcCount

ApplGgdaDisableOnChangeDtcCount

Prototype	
void **ApplGgdaDisableOnChangeDtcCount** (GgdaContextIndexType context)	
Parameter	
- context	- Identifies the channel for which the counting is to be cancelled. You can safely ignore this parameter if the GGDA only handles one channel.
Return code	
-	-
Functional Description	
This function will be called to notify the application for stopping the background DTC count monitoring mechanism.	
Particularities and Limitations	
■ Available only if mode $A9 is enabled and level $82 is supported	
Call context	
■ Called from the **GgdaStateTask** context.	

Table 4-16   ApplGgdaDisableOnChangeDtcCount

GgdaRdiOnDtcCountChanged

Prototype	
`void ` **`GgdaRdiOnDtcCountChanged`** ` (GgdaContextIndexType context,` `                              vuint16 newCount)`	
Parameter	
- context - newCount	- Identifies the channel for which the request is answered. Please pass '0' (zero) here if the GGDA only handles one channel. - (Changed) new number of DTCs
Return code	
-	-
Functional Description	
Call this function when you have detected a DTC count change.	
Particularities and Limitations	
■ The application is responsible to support several concurrent counting algorithms. Requests on different channels may request different status mask ■ Available only if mode $A9 is enabled and level $82 is supported	
Call context	
■ Any.	

Table 4-17   GgdaRdiOnDtcCountChanged

GgdaRdiDeactivateOnChangeDtcCount

Prototype	
`void ` **`GgdaRdiDeactivateOnChangeDtcCount`** ` ( GgdaContextIndexType context)`	
Parameter	
- context	- Identifies the channel for which the counting is to be deactivated. Please pass '0' (zero) here if the GGDA only handles one channel.
Return code	
-	-
Functional Description	
Call this function if you need explicitly to stop the DTC count change activity in the GGDA component.	
Particularities and Limitations	
■ Available only if mode $A9 is enabled and level $82 is supported	
Call context	
■ Any.	

Table 4-18   GgdaRdiDeactivateOnChangeDtcCount

5   Configuration in CANGen

Some of the configuration aspects were already mentioned at the affected features of the component. The remaining set of configuration options is described here:

5.1   Communication Parameter
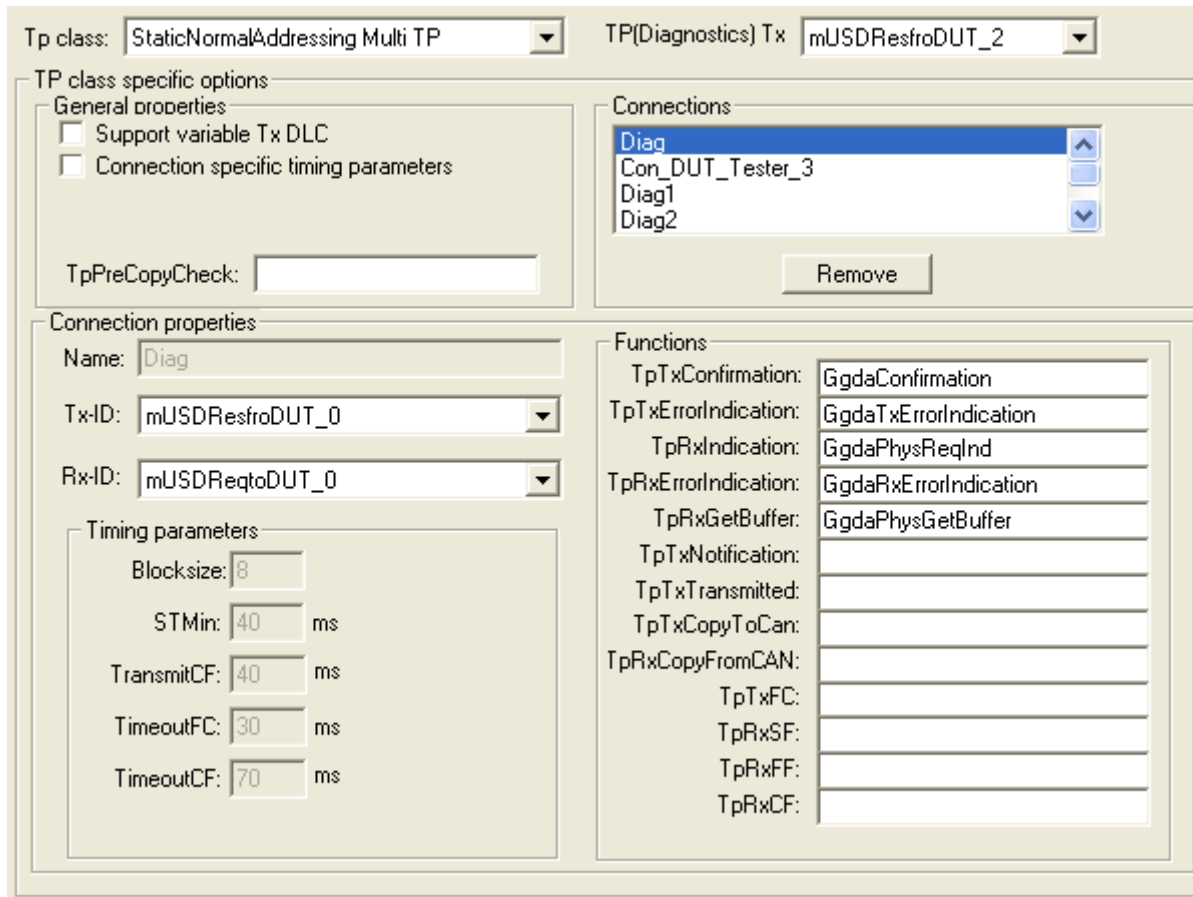
5.1.1   1st step – GENtool

Prior adding your database files in your configuration you shall assure the following requirements:

- The CAN channel on which CANdesc shall operate (HS-CAN) shall use a DBC file with all CANdesc required message attributes (for USDT and UUDT messages) set to the appropriate values. See [1] for further instructions.

- For CANgen versions prior to 4.23.49, the UUDT messages used by the GGDA may **not** have the message type attribute set in the DBC file.

5.1.1.1   USDT Connection Configuration

Once you have prepared the DBC files and their CAN mapping, you can create your configuration as usual. Since CANdesc is automatically prepared for the TPMC API you have only to adapt the hook functions for each GGDA TP connection:

Figure 5-1 GGDA OSEK-TP configuration

The TP timing parameters are the same as those for the CANdesc connection.

> **Info**
> Please note that this is an example configuration where the GGDA was configured to use the connections 0 and 2. The connection numbers may vary for your configuration.

The transport layer must be configured to support an additional feature for the GGDA component: **GM requires that any request length (except zero-length) shall be accepted independent of the buffer size. After the reception has been accomplished with success, it shall be checked if the buffer size is enough to handle the whole request data. If it is not true, a negative response $12 must be sent.**

To activate the buffer-overflow option of the TPMC, used for implementing this requirement, a user-config file is needed to be prepared with the following content:

```
#define TP_USE_OVERRUN_INDICATION kTpOn
```

Assuming that the file name was **tpmc_ggda.cfg**, you have to add the path to it in the GENtool as shown below:
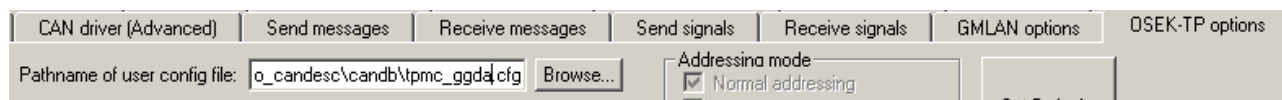
Figure 5-2 User-config file for the GGDA TPMC configuration.

A functionally requested service is not received by the TPMC, but by the GGDA component it self. Therefore the necessary pre-copy function must be configured manually in the GENtool as shown below:

Figure 5-3 „GgdaFuncPrecopy" configuration.

> **Caution**
> If you have the 'DiagState' attribute set to 'true' in your database for the GGDA channels, you need to also select the 'App-Message' check box.
> Else, the 'GgdaFuncPrecopy' function will be replaced by 'TpFuncPrecopy' during the generation process.

5.1.1.2 UUDT Transmitter Configuration

Since the GGDA component handles also the $A9 service, the embedded UUDT transmitter must be configured too. Please, enter the confirmation function name for the UUDT message on the GGDA CAN channel as shown below (channel 2 messages not visible here):

Message		PreTransmit Func	Confirmation Func	Properties	
0x00614000	mE010VN02_0				...
0x00610000	mCE008VN01_0				...
0x0060C000	mC006VN03_0				...
0x00608000	mE004VN01_0				...
0x00606000	mC003VN02_0				...
0x00604000	mE002VN01_0				...
0x00602000	mC001VN01_0				...
0x661	USDT_ApplResFromDUT				...
0x644	mUSDResfroDUT_0		TpDrvTxConfirmation		...
0x624	mDUTNetMan_0	NmPreTransmit	NmConfirmation		...
0x544	mUUDResfroDUT_0		GgdaUudtConfirmation		...
0x00614000	mE010VN02_1				...
0x00610000	mCE008VN01_1				...
0x0060C000	mC006VN03_1				...
0x00608000	mE004VN01_1				...
0x00606000	mC003VN02_1				...
0x00604000	mE002VN01_1				...
0x00602000	mC001VN01_1				...
0x644	mUSDResfroDUT_1		TpDrvTxConfirmation		...
0x624	mDUTNetMan_1	NmPreTransmit	NmConfirmation		...
0x544	mUUDResfroDUT_1		DescUudtConfirmation		...

Figure 5-4 „GgdaUudtConfirmation" configuration

> **Caution**
> From CANGen 4.23.49 on, please select the 'App-Message' check box for the UUDT message the GGDA is to use.

5.1.2   2nd step – Ggda_par.h file

Once you are done with the GENtool settings, you have to adapt the GGDA module configuration in the Ggda_par.h file. Most settings default to the same value as configured for the CANdesc component, but they can be changed if needed.

The number of channels the GGDA works on defaults to the number of CAN channels minus one, for the full CANdesc diagnostics. This can be changed in Ggda_par.h:

```
#define kGgdaNumContexts                    (kCanNumberOfChannels - 1)
```

5.1.3   3rd step – Ggda_par.c file

In the Ggda_par.c file the connection parameters are configured.

The first step is change the following #include directive to include your configuration header file, and to remove the #error directive:

```
#error "Please include your ECU configuration header file. Then remove this line"
#include "YourECU.h"
```

5.1.3.1   CAN Channel Configuration

For each channel you need to fill out a configuration context struct in the configuration array GgdaCanConfig.

```
typedef
{
#if (GGDA_CONFIG_SERVICE_A9_SUPPORT == GGDA_CONFIG_ON)
   CanTransmitHandle        uudtResMsgHandle;
#endif
   CanChannelHandle         canChannel;
   canuint8                 optionalModes;
}GgdaCanConfigType;
```
The optional modes each channel can support are

```
kGgdaDisableAll          No optional modes supported
kGgdaEnableModeA502      Enable HiSpeed support (mode $A5 02)
kGgdaEnableMode1002      Enable support for DisableAllDTCs request (mode $10 02)
kGgdaEnableMode1004      Enable support for WakeUpLink request (mode $10 04)
kGgdaEnableMode10xx      Enable support for all levels of mode $10
kGgdaEnableModeA980      Enable support for mode $A9 80 and $A9 81
kGgdaEnableModeA981      Enable support for mode $A9 81
kGgdaEnableModeA982      Enable support for mode $A9 81 and $A9 82
kGgdaEnableModeA9xx      Enable support for all levels of mode $A9
kGgdaEnableAll           Enable all optional modes
```

⚠	**Caution** Please take into account that the settings in this table must be consistent with those that determine the availability of the corresponding service. You have to set the appropriate service specific setting to have consistent configuration on GGDA (please refer chapters : *5.3 "Supported Diagnostic Services"*).

Example

🔳	**Example** For a configuration with 3 channels, CAN0 and CAN2 are handled by GGDA. The CAN message for UUD responses is named mUUDResfroDut, and CAN0 supports HiSpeed programming. Both channels shall support modes $10 02 and $10 04. The configuration would look like this:

```
const GgdaCanConfigType ggdaCanConfig[kGgdaNumContexts] =
{
    {
#if (GGDA_CONFIG_SERVICE_A9_SUPPORT == GGDA_CONFIG_ON)
        CanTxmUUDResfroDUT_0,            /* UudtResMsgHandle for Can 0*/
#endif
        kCanIndex0,                      /* CanChannel 0 */
        kGgdaEnableAll                   /* Levels 02, 04 - Hispeed support */
    },
    {
#if (GGDA_CONFIG_SERVICE_A9_SUPPORT == GGDA_CONFIG_ON)
        CanTxmUUDResfroDUT_2,            /* UudtResMsgHandle for Can 2*/
#endif
        kCanIndex2,                      /* CanChannel for Can 0 */
        kGgdaEnable1002 | kGgdaEnable1004 /* Levels 02, 04 - No HiSpeed support */
    }
};
```

⚠	**Caution** Please take into account that the message transmission handle is configuration dependent (refer the section *5.3.2 "ReadDiagnosticInformation ($A9)"*!

5.1.3.2 Static TP Channel Configuration

The static TP channel configuration is separate from the CAN configuration in the context configuration array `ggdaTpConfig`. Again, a struct needs to be filled in for each channel.

```
typedef struct
{
    canuint8 tpTxChannel;
    canuint8 tpRxChannel;
}GgdaTpConfigType;
```

Example

> **Example**
> For the example the TP is configured as described in chapter 5.1.1.1. The CAN transmit/receive handles are prefixed with "CanTx"/"CanRx".
>
> ```
> const GgdaTpConfigType ggdaTpConfig[kGgdaNumContexts] =
> {
>    /* Tp channels for Can 0 */
>    {
>       CanTxDiag,            /* TpTxChannel for Can 0*/
>       CanRxDiag             /* TpRxChannel for Can 0*/
>    },
>    /* Tp channels for Can 2 */
>    {
>       CanTxDiag2,           /* TpTxChannel for Can 2*/
>       CanRxDiag2            /* TpRxChannel for Can 2*/
>    }
> };
> ```

> **Caution**
> Please take care and assure that the order of ggdaCanConfig and ggdaTpConfig is the same.

> **Info**
> The context parameter passed to the context depending callback functions ApplGgda* actually is an index into these configuration arrays. If you need more information about the CAN channel on which a request was received, you can include ggda_par.h in your application and access these configuration structures.

5.2   Timings Parameter

All needed GM diagnostics time parameters are taken from the CANdesc configuration and should therefore comply with GMW 3110 v 1.5, but they are still configurable. If you need to adjust a parameter, you can adapt the GGDA module configuration in the file Ggda_par.h (all times are in ms units):

```
#define kGgdaTimeoutS1                     5000 /*ms*/
```

```
#define kGgdaTimeoutP2                     100  /*ms*/
#define kGgdaTimeoutP2Ex                   1000 /*ms*/
```

```
#define kGgdaTimeoutVnDiagnostics          8000 /*ms*/
```

5.3 Supported Diagnostic Services

Some of the services implemented in GGDA are optionally supported (depends on the current ECU specification). Therefore they can be deactivated in order to save ECU resources.

5.3.1 InitializeDiagnosticOperationMode ($10 $xx)

In order to specify whether any sub-function of this service will be supported or not by GGDA, you have to modify the switches in the Ggda_par.h file. Example for activating the services:

#define GGDA_CONFIG_SERVICE_10_02_SUPPORT	GGDA_CONFIG_ON

#define GGDA_CONFIG_SERVICE_10_04_SUPPORT	GGDA_CONFIG_ON

> ⚠ **Caution**
> Please take into account that this switch enables/disables the complete support of this service. You have to set the appropriate channel specific setting to have consistent configuration on GGDA (please refer *5.1.3.1"CAN Channel Configuration"*).

5.3.2 ReadDiagnosticInformation ($A9)

In order to specify whether this service will be supported or not by GGDA, you have to modify the switch in the Ggda_par.h file. Example for activating the service:

#define GGDA_CONFIG_SERVICE_A9_SUPPORT	GGDA_CONFIG_ON
#define GGDA_CONFIG_SERVICE_A9_80_SUPPORT	GGDA_CONFIG_ON
#define GGDA_CONFIG_SERVICE_A9_82_SUPPORT	GGDA_CONFIG_ON

Mode $A9 81 is always enabled if GGDA_CONFIG_SERVICE_A9_SUPPORT is set to GGDA_CONFIG_ON.

5.3.3 ReadDiagnosticInformation ($A5 $02)

In order to specify whether this service will be supported or not by GGDA, you have to modify the switch in the Ggda_par.h file. Example for activating the service:

#define GGDA_CONFIG_SERVICE_A5_02_SUPPORT	GGDA_CONFIG_ON

> ⚠ **Caution**
> Please take into account that this switch enables/disables the complete support of this service. You have to set the appropriate channel specific setting to have consistent configuration on GGDA (please refer *5.1.3.1"CAN Channel Configuration"*).

5.4   Development and Integration Support

In ordert to lighten the integration support of GGDA in your ECU this module provide own monitoring functionality than can be optionally turned one using the switch in Ggda_par.h. Example for turning debug support on:

```
#define GGDA_CONFIG_DEBUG              GGDA_CONFIG_ON
```

5.5   Target Address Acceptance on Functional Requests

GGDA handles by itself the functional request reception. The used target address will be evaluated and GGDA decides whether to accept or ignore the requested frame. There are two use-cases for the ECUs: a non-gateway and a gateway ECUs. GGDA synchronizes the current use case with the CANdesc use case detection (please refer the [1] for more details on CANdesc gateway usage configuration).

If you want to override this detection use the switch below to specify your use-case:

```
#define GGDA_CONFIG_GW_SUPPORT              GGDA_CONFIG_ON
```

6   Configuration in GENy

6.1   Communication Parameters

For all channels the GGDA shall be used on set up the communication databases as described in [1]

The activate the Diag_CanDescGgdaExt_Gm module in the GENy tool on the appropriate channels. You will not be able to activate the Ggda module on the same channel as CANdesc.

If the DBC file is set up correctly the communication configuration will be set up automatically.

6.2   General Parameters

You can configure the parameters that affect GGDA as a whole on the 'root' Diag_CanDescGgdaExt_Gm entry of the component tree.

Figure 6-1 Global configuration parameters in GENy

ECU settings, Timing Parameters
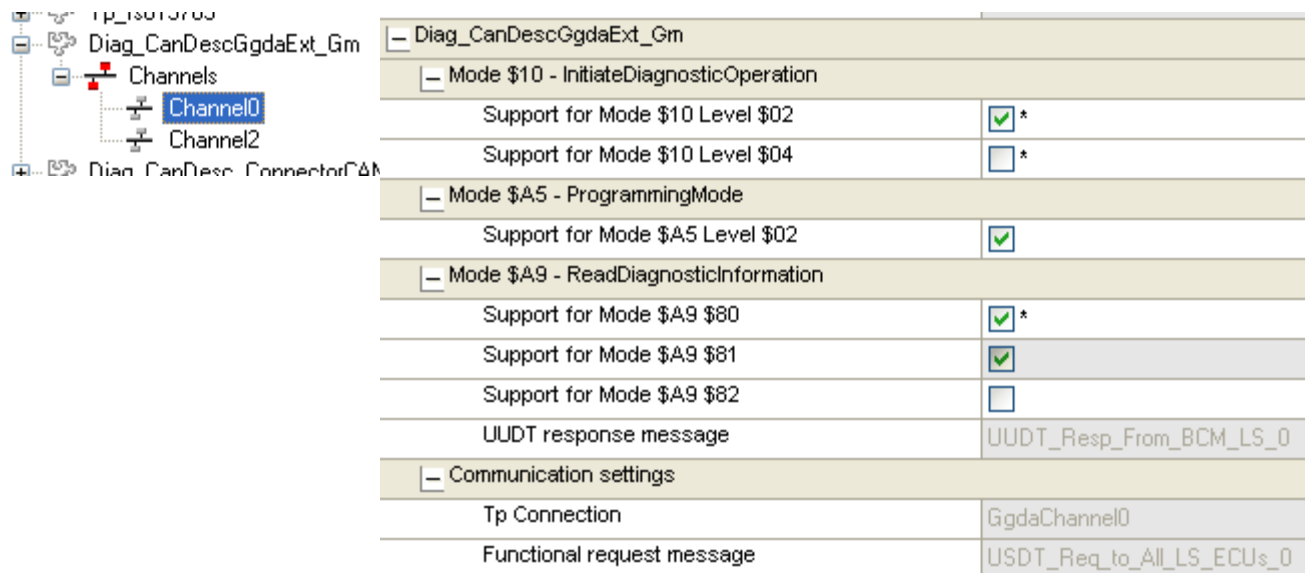
Most parameters default to the CANdesc values, which is the recommended setting. If you disable the checkbox 'Use CANdesc setting', you are able to override the defaults with own values.

6.3   Supported Diagnostic services

You can configure the channel specific parameters depending on the channel entries below the Diag_CanDescGgdaExt_Gm entry.

The HiSpeed support is only available if that functionality is supported by the Nm_Gmlan_Gm module. To allow more flexibility in use cases, the configuration tool does not prevent to enable support for mode $A5 02 on a DW Link, and does not enforce it on a SW Link either.

If mode $A9 is supported, the level $81 is mandatory and cannot be disabled.

Figure 6-2 Channel specific configuration parameters in GENy

7 Integration

In order to integrate properly GGDA, your diagnostic application must comply to the following include rules in this order:

- Always include **can_inc.h** (resp. **v_inc.h**) as first header file.
- If this application source file manages the TPMC initialization and task calls, include the **tpmc.h** file;
- Include **desc.h**;
- Include **ggda.h**;
- Include (if necessary) all header files required by your application source file.

	Caution
⚠	Using CANGen, the GGDA is not supported by CCL. In this case your application needs to explicitly call GgdaInitPowerOn, GgdaStateTask and GgdaTimerTask. Using GENy this restriction does not apply.

8 Contact

Visit our website for more information on

> News

> Products

> Demo software

> Support

> Training data

> Addresses

www.vector.com