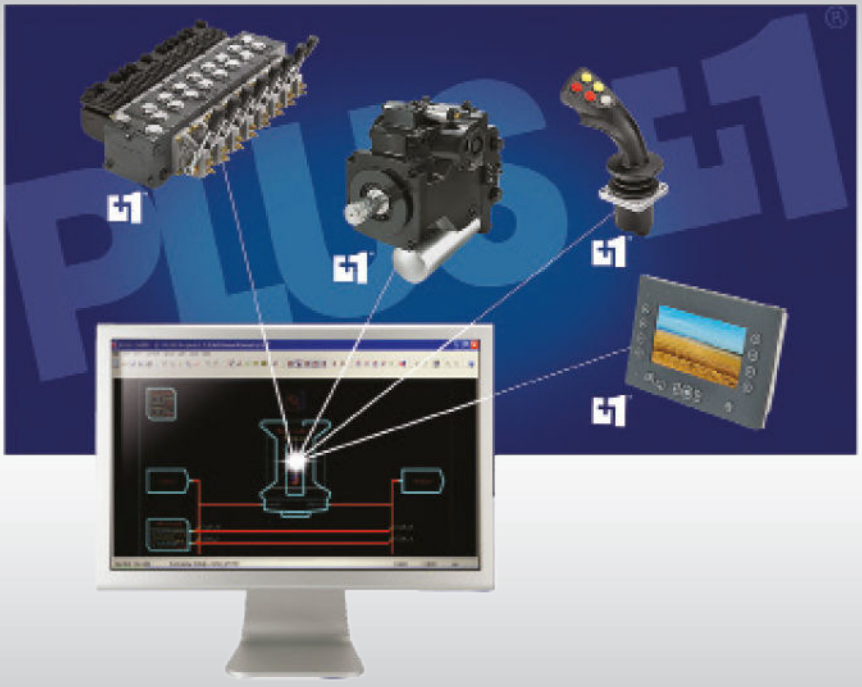
MAKING MODERN LIVING POSSIBLE

User Manual

PLUS+1®
Diagnostic API



Revision history

Table of revisions

Date	Changed	Rev
October 2015	Various updates; Conversion to Danfoss layout	0100
March 2011	First edition; PLUS+1® Diagnostic API release 5.0	0000

Contents

Contents

General information

About this manual

This document describes PLUS+1® Diagnostic API which allows diagnostics of PLUS+1® hardware using CAN interface. Included in this document is the *RP1210A requirements for PLUS+1® diagnostics*.

Overview

The PLUS+1® Diagnostic API allows diagnostics of a PLUS+1® hardware using a CAN interface. The key functionality includes:

- Log parameters
- Write parameters
- Download applications

The API is implemented as a Windows® Dynamic Link Library (DLL) for use under Microsoft® Windows® 7 32-bit/64-bit and Windows® 8/8.1 64-bit.

ECU application support

- PLUS+1® controller, display, Firmware GUIDE applications with ToolKey enabled. PLUS+C Firmware, controller and joystick applications.
- OS using InfoBlock 9 or 11 is required.
  - ECU Boot loader must support application baud rate.

CAN interface support

The API supports the CG150, Kvaser USBcanII, RP1210A compliant CAN interface. Reference *RP1210A requirements for PLUS+1® diagnostics*.

PLUS+1® GUIDE Drivers version 2.0.6 or later needs to be installed on the client PC to use this product with CG150/CG150-2 CAN interfaces.

Reserved CAN ID ranges

The following extended 29-bit CAN ID range is used by PLUS+1® protocol: 0xCDA0000-0xCDBFFFF

License

A PLUS+1® Diagnostic API license is required to run the API. This license is unique for each customer. The DLL key for the particular license needs to be known when implementing the API.

Unlike a PLUS+1® Service Tool license the API license is not locked to a specific PC and does not have an expiration date.

A PLUS+1® Service Tool license will not work with the API.

User traceability

The PLUS+1® Diagnostic API has the same traceability features as the PLUS+1® Service Tool.

The following information is stored in the controller:

Who	The license ID and short name or Customer ID depending on license.
When	Date actions were performed.
What	What actions were performed.

General information

The following activities are stored in the controller:

- Downloading an application or read only parameter file to a controller
- Adjusting of parameters
- Reading variables

API design

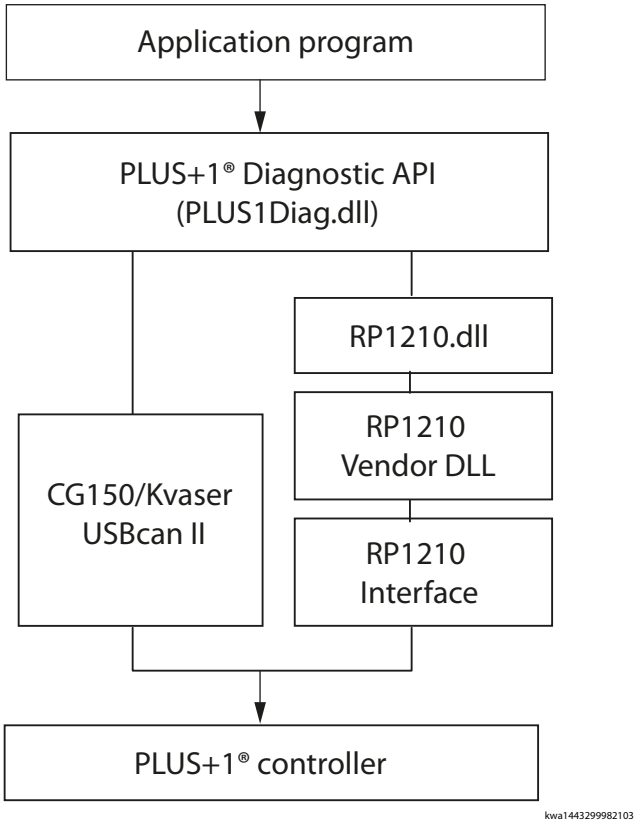
API files

File	Description
Plus1diag.dll	The API DLL that is called from the application
RP1210.dll	Interface DLL for handling the RP1210A calls
danfosscanlib32.dll	CG150/CG150-2
danfosskvalapw2.dll	CG150/CG150-2

Calling convention

All functions use the stdcall calling convention.

Architectural design



High-level design

- The API provides a set of high-level functions to allow diagnostics of PLUS+1® hardware.
- All API functions are blocking and will return when the operation is complete.

Usage scenarios

General

All functions in this API return a signed 4 byte integer. -1 indicates success for any function.

This API is not thread safe, so never call another function before the previous one has returned. The only exception is: `P1_DownloadCancel` should only be called after `P1_DownloadApplication` was called and before it returns.

Program startup

1. Normal program initialization code.
2. Load in the DLL Plus1diag.dll dynamically (use LoadLibrary).
3. Load all functions described in this document (use GetProcAddress). If any functions will not load, the DLL may not be up to date or there is a newer version of *PLUS+1® Diagnostic API User Manual*, **L1028305**.

Going online

1. Decide what gateway, channel and baudrate etc you want to use.
2. Call `P1_Open` using the values from the previous step, make sure it returns -1, 2. otherwise check *Return codes* on page 51.

Obtaining node information

1. Call `P1_ScanSystem`, check that the return code is -1.
2. If the return code was -1 you can see how many nodes are present. For each node:
   a. Call `P1_GetNodeAddress` to convert the node index to a node address.
   b. Call `P1_LoadNodeData` to get node data from the node.
   c. Call `P1_GetNodeInfo` 16 times (once for each item) to extract some information about the node. This information can later be shown to the user.
   d. Call `P1_GetParameterCount` to see how many parameters the node has.
   e. For each parameter: Call `P1_GetParameterData` 6 times (once for each item) to extract information about the parameter. This information can later be shown to the user.

Reading unit history

1. For each node:
   a. Call `P1_ReadUnitHistory`, check that the return code is -1.
   b. Call `P1_GetUnitHistoryData` 7 times, once for each item.

Reading single parameter

1. Call `P1_ReadParameter`, check that the return code is -1.

Usage scenarios

Reading multiple parameters

1. Call `P1_ClearReadParameterList`, check that the return code is -1.
2. For each parameter you want to read:
   a. Call `P1_AddReadParameterList`, check that the return code is -1.

   Important: you must remember what parameters you add, and in what order.
3. Call `P1_ReadParameterList`, check that the return code is -1. Parse the string returned 3. for values given in same order as you added the parameters.

Writing single parameter

1. If `P1_GetParameterData` was called then information has been received regarding if 1. parameter is writable. (See *Obtaining node information* on page 8, sequence 2e).
2. If writable, call `P1_WriteParameter`, check that the return code is -1.

Writing multiple parameters

1. Call `P1_ClearWriteParameterList`, check that the return code is -1.
2. For each parameter you want to write do:
   a. If `P1_GetParameterData` was called then information has been received regarding if parameter is writable. (See *Obtaining node information* on page 8, sequence 2e).
   b. If writable, call `P1_AddWriteParameterList`, check that the return code is -1.

   Important: you must remember what parameters you add, and in what order.
3. Call `P1_ReadParameterList`, check that the return code is -1. Parse the string returned for individual error codes given in same order as you added the parameter.

Downloading application

1. Call `P1_OpenApplication`, specifying node and file, check that the return code is -1.
2. Call `P1_DownloadApplication`, check that the return code is -1. The progress 2. function is optional, but keep in mind that this function will often take a couple of minutes before return.
3. After download you must repeat the sequence of steps of *Obtaining node information* on page 8.

Checking node presence on bus

1. Read 1 parameter from each node you found in sequence of steps of *Obtaining node information* on page 8. (If parameter can't be read 3 times in a row, the node is probably offline).

Going offline

1. If you have successfully called `P1_Open`, but not `P1_Close`, then call `P1_Close`.

Program shutdown

1. Normal program shutdown code.

PLUS+1® diagnostic base API functions

P1_Open

C++ Syntax

```
int P1_Open(char* DllKey,
    char* LicenseKey,
    unsigned short Gateway,
    char* GatewayDLL,
    unsigned int Baudrate,
    unsigned short DeviceID,
    char* Options, char* DiagnosticFilePath);
```

Delphi Syntax

```
function P1_Open(DllKey : PAnsiChar;
    LicenseKey : PAnsiChar;
    Gateway : Word;
    GatewayDLL : PAnsiChar;
    Baudrate : Longword;
    DeviceID : Word;
    Options : PAnsiChar;
    DiagnosticFilePath : PAnsiChar) : Integer;
```

Description

This function will establish a connection with the API. Inside the API DLL, the function will allocate data structures and initialize the CAN interface driver. The API license will also be verified and check with the DLL key. The license is required to run this function before running any of the following API functions.

Gateway Error can only be detected by using the `P1_Open` function. If the gateway has been disconnected from the PC the `P1_Close` needs to be called followed by `P1_Open`.

Input parameters

Parameter	Description	
DllKey	API DLL key	
LicenseKey	API license key	
Gateway	1 = KvaserUSBcan II	
	2 = RP1210A	
	3 = CG150/CG150-2 (Requires danfosscanlib32.dll and danfosskvalapw2.dll)	
GatewayDLL	Fullpath to the RP1210A vendor DLL (This parameter is only required when using RP1210A)	
Baudrate*	CAN bus bit rate (Measured in bits per second)	
	Supported values	
	50000	50 kbit/s
	100000	100 kbit/s
	125000	125 kbit/s
	250000	250 kbit/s (default)
	500000	500 kbit/s
	1000000	1 Mbit/s
DeviceID**	The device id or channel number starting at 0 (zero)	

PLUS+1® diagnostic base API functions

Parameter	Description		
Options***	Configuration keywords passed as a string. Keywords should be followed by an equal sign and the value enclosed in double-quotes ("). Different keywords should be separated with a space character.		
	Supported keywords	**Description**	**Default values**
	KWP2000_AddTimeOut	Milliseconds timeout uses in the extended KWP2000 protocol.	0
	KWP2000_STmin	See ISO 15765-2 for STmin (Separation time minimum).	0
	KWP2000_BlockSize	See ISO 15765-2 for BS (BlockSize).	0
	RP1210_Protocol	The fpchProtocol parameter passed to the function RP1210_ClientConnect.	CAN
	RP1210_BlockOnRead	Set to 1 to set the BlockOnRead parameter in RP1210_ReadMessage.	0
	RP1210_BlockOnSend	Set to 1 to set the BlockOnSend parameter in RP1210_SendMessage.	0
	RP1210_ReadBigEndian	Set to 0 to enable little endian addressing in RP1210_ReadMessage.	1
	RP1210_SendBigEndian	Set to 0 to enable little endian addressing in RP1210_SendMessage.	1
	Example for enabling BlockOnRead and BlockOnSend: RP1210_BlockOnRead ="1" RP1210_BlockOnSend="1"		
DiagnosticFilePath	File path to the directory where diagnostic data, uploaded from the node, is stored.		

* This parameter is not used when using RP1210A.

** This is the DeviceID specified in the RP1210 vendor INI File when using RP1210A.

*** Keywords are optional and can be omitted if the default value is correct.

Return value

Value	Description
-1	Success

User errors

Error	Description
205	Invalid baudrate
500	API already started
504	Incorrect APIDLL filename
507	Invalid gateway selection
508	Gateway driver not found
509	DiagnosticFilePath not found
522	Gateway Error
533	Invalid option value
554	Diagnostic Data file already installed

PLUS+1® diagnostic base API functions

License errors

Error	Description
161	Invalid API license key

P1_Close

C++ Syntax

```
int P1_Close( );
```

Delphi Syntax

```
function P1_Close : Integer;
```

Description

This function closes the connection with the API and de-allocates internal data structures and closes the connection with the CAN interface. It is required to run `P1_Close` before the application if a connection has been made with `P1_Open`.

Return value

Value	Description
-1	Success

User errors

Error	Description
501	API not started

P1_SetCustomerID

C++ Syntax

```
int P1_SetCustomerID(unsigned int CustomerID);
```

Delphi Syntax

```
function P1_SetCustomerID(CustomerID : Cardinal) : Integer;
```

Description

This function sets the customer ID number to use in the unit history. CustomerID can be in the range of 0 to 32767 (0..7FFF hex).

The CustomerID feature must be enabled in the API license for this function to work.

Input parameters

Parameters	Description
CustomerID	User identity used in the unit history

PLUS+1® diagnostic base API functions

Return value

Value	Description
-1	Success

User errors

Error	Description
501	API not started
510	License does not support Customer ID
511	Invalid Customer ID value

P1_ScanSystem

C++ Syntax

```
int P1_ScanSystem(char* ToolKey,
    unsigned short* NodeCount);
```

Delphi Syntax

```
function P1_ScanSystem(ToolKey : PAnsiChar;
    var NodeCount : Word) : Integer;
```

Description

This function scans the CAN bus and returns the number of found nodes.

Input parameters

Parameter	Description
ToolKey*	Toolkey is used for unlocking the application and all applications in the system. Different toolkeys are not supported in the system.

* ToolKey is only available for PLUS+1® GUIDE applications

Output parameters

Parameter	Description
NodeCount	A pointer to a DWORD which will receive the total number of found nodes.

Return value

Value	Description
-1	Success

User errors

Error	Description
200	Error loading gateway drivers
201	API function already running
204	No answer from node
501	API not started

PLUS+1® diagnostic base API functions

P1_GetNodeAddress

C++ Syntax

```
int P1_GetNodeAddress(unsigned short NodeIndex;
    unsigned short* NodeAddress);
```

Delphi Syntax

```
function P1_GetNodeAddress(NodeIndex : Word;
    var NodeAddress : Word) : Integer;
```

Description

This function returns the node address for a node.

Input parameters

Parameter	Description
NodeIndex	Number of nodes to retrieve the node address, beginning at 0 (zero) and ending at the NodeCount value returned from the function `P1_ScanSystem` minus 1.

Output parameters

Parameter	Description
NodeAddress	A pointer to a DWORD which will receive the node address. The most significant byte is the net number and the least significant byte is the node number.

Return value

Value	Description
-1	Success

User errors

Error	Description
501	API not started
512	No nodes found
513	Invalid node index

P1_LoadNodeData

C++ Syntax

```
typedef TP1_ProgressFunc(unsigned short PercentCompleted);
```

```
int P1_LoadNodeData(unsigned short NodeAddress;
    TP1_ProgressFunc ProgressProc);
```

PLUS+1® diagnostic base API functions

Delphi Syntax

```
function P1_LoadNodeData(NodeAddress : Word;
    ProgressProc : TFarProc) : Integer;
```

Description

This function loads the diagnostic data from a node. The application is unlocked using the toolkey supplied in `P1_ScanSystem`.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
ProgressProc	Callback function that gives feedback on the progress (0 to 100 percent). A null pointer can optionally be passed in if the callback functionality is undesired.

Return value

Value	Description
-1	Success

Warnings

Warning	Description
215	No valid diagnostic data available
268	Access denied to application without toolkey
558	ECU is in boot loader mode. Download a valid ECU application.

File download allowed, but no access to parameter read/write functions.

User errors

Error	Description
201	API function already running
202	Invalid node address
204	No answer from node
242	Incorrect toolkey
265	Security access timeout
501	API not started
518	Access to node denied

ECU errors

Error	Description
166	Unknown key value
211	Unsupported infoblock
212	Unable to load diagnostic data
213	Unable to restore diagnostic data
214	Unable to read diagnostic data
238	Infoblock checksum error

PLUS+1® diagnostic base API functions

Error	Description
241	Unable to read node data
248	Security access denied
249	Unable to start diagnostic session
250	Invalid diagnostic data length
271	Diagnostic Data Error: Index out of range
272	Diagnostic Data Error: Invalid address vector length
273	Diagnostic Data Error: No end tag
274	Diagnostic Data Error: Signal outside struct group
275	Diagnostic Data Error: Invalid data type
276	Diagnostic Data Error:Invalid flag
277	Diagnostic Data Error: Invalid struct format
278	Diagnostic Data Error: Invalid struct group format
279	Diagnostic Data Error: Invalid array range
280	Diagnostic Data Error: Invalid end of data
517	Unable to read node type
519	Unable to read unit history

P1_GetNodeInfo

C++ Syntax

```
int P1_GetNodeInfo(unsigned short NodeAddress,
    unsigned short Item,
    void *Buffer,
    unsigned int BufSize);
```

Delphi Syntax

```
function P1_GetNodeInfo(NodeAddress : Word;
    Item : Word;
    var Buffer;
    BufSize : Cardinal) : Integer;
```

Description

This function retrieves node information read by the `P1_LoadNodeData` function.

PLUS+1® diagnostic base API functions

Input parameters

Parameter	Description	
NodeAddressItem*	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.	
	0 = **Application ID**	
	1 = **Application Type** (only for GUIDE applications)	
	2 = **Application Version** (only for GUIDE applications)	
	3 = **Node Type**	0 = Controller Programmable
		1 = Firmware
		3 = Joystick
		6 = Display Programmable
		9 = Controller
		10 = Display
	4 = **GUIDE version**	
	5 = **OS string**	
	6 = **GUIDE compile date/time**	
	7 = **Boot loader version**	
	8 = **EAN**	
	9 = **Serial number**	
	10 = **Birthdate**	
	11 = **Part number 0**	
	12 = **Part number 1**	
	13 = **Part number revision 0**	
	14 = **Part number revision 1**	
	15 = **Programmable status**	-1 = Undefined
		0 = Non-programmable
		1 = PLUS C programmable
		2 = GUIDE programmable
	16 = **Part number 2**	
	17 = **Application release status**	0 = Released
		1 = Not released (application created using an unreleased version of PLUS+1® GUIDE or HWD).
	18 = **SIL2 Certified**	0 = NotSIL2certified
		1 = SIL2certified
	19 = **Diagnostic Data filename**	
	20 = **Global Read Access Right**	D = Not used in this ECU
		0-9 = Read Access Right used for this ECU
	21 = **Global Write Access Right**	D = Not used in this ECU
		0-9 = Write Access Right used for this ECU
	22 = Name of tool used to develop the application (PLUS+1® GUIDE or PLUS+C)	
	23 = **Software mode**	-1 = Undefined
		0 = In application
		1 = In boot loader
	24 = **ECU history supported**	0 = Not supported
		1 = Supported
	25 = **System ID**	

* All data is returned as null-terminated character strings.

PLUS+1® diagnostic base API functions

Parameter	Description
Buffer	The size of the buffer to which the buffer points.

Output parameters

Parameter	Description
Buffer	The address of a buffer which is to receive the data.

Return value

Value	Description
-1	Success

User errors

Error	Description
202	Invalid node address
501	API not started
502	Diagnostic data not loaded
514	Invalid Item value
516	Buffer to small

P1_ReadUnitHistory

C++ Syntax

```
int P1_ReadUnitHistory(unsigned short NodeAddress,
    unsigned short* RecordCount);
```

Delphi Syntax

```
function P1_ReadUnitHistory(NodeAddress : Word;
    var RecordCount : Word) : Integer;
```

Description

This function reads the unit history from a node. The unit history information can be read out with `P1_GetUnitHistoryData`.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.

Output parameters

Parameter	Description
RecordCount	A pointer to a DWORD which will receive total number of unit history records.

PLUS+1® diagnostic base API functions

Return value

Value	Description
-1	Success

User errors

Error	Description
201	API function already running
202	Invalid node address
204	No answer from node
501	API not started
502	Diagnostic data not loaded
526	Access denied to function

ECU errors

Error	Description
203	Invalid answer
248	Security access denied

P1_GetUnitHistoryData

C++ Syntax

```
int P1_GetUnitHistoryData(unsigned short NodeAddress,
    unsigned short Index,
    unsigned short Item,
    char* Buffer,
    unsigned int BufSize);
```

Delphi Syntax

```
function P1_GetUnitHistoryData(NodeAddress : Word;
    Index : Word;
    Item : Word;
    var Buffer;
    BufSize : Cardinal) : Integer;
```

Description

This function returns one record of the unit history. It is necessary to run this function repeatedly with different Item values to read out a complete unit history record.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
Index	The number of the unit history record to retrieve information, beginning at 0 (zero) and ending at the RecordCount value returned from the function `P1_ReadUnitHistory` minus 1.

PLUS+1® diagnostic base API functions

Parameter	Description	
Item *	Selects what type of data to retrieve.	
	0 = **Activity**	0 = Application Download
		1 = Read Only Parameter Download
		2 = Parameter Write
		3 = Parameter Read
	1 = **Date** (YYYYMMDD)	
	2 = **License ID** (only returns a ID if the record was created by a different PLUS+1® GUIDE user)	
	3 = **Shortname** (2 characters, only returns a Shortname if the record was created by a different PLUS+1® GUIDE user)	
	4 = **Customer ID** (only returns a ID if Customer ID is enabled in the API license and if the record was created with the same API license)	
	5 = **TimeKey**	
	6 = **Downloaded file** (First 8 characters of filename) Only available for Download records	
BufSize	The size of the buffer to which the buffer points.	

* All data is returned as null-terminated character strings.

Output parameters

Parameter	Description
Buffer	The address of a buffer which is to receive the data.

Return value

Value	Description
-1	Success

User errors

Error	Description
202	Invalid node address
501	API not started
502	Diagnostic data not loaded
514	Invalid Item value
515	Invalid Index value
516	Buffer to small
526	Access denied to function

P1_GetParameterCount

C++ Syntax

```
int P1_GetParameterCount(unsigned short NodeAddress,
    unsigned int* ParameterCount);
```

PLUS+1® diagnostic base API functions

Delphi Syntax

```
function P1_GetParameterCount(NodeAddress : Word;
    var ParameterCount : Cardinal) : Integer;
```

Description

This function retrieves the total number of parameters for a node.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.

Output parameters

Parameter	Description
ParameterCount	A pointer to a LONG which will receive total number of parameters.

Return value

Value	Description
-1	Success

User errors

Error	Description
202	Invalid node address
501	API not started
502	Diagnostic data not loaded

P1_GetParameterData

C++ Syntax

```
int P1_GetParameterData(unsigned short NodeAddress,
    unsigned int Index,
    unsigned short Item,
    char* Buffer,
    unsigned int BufSize);
```

Delphi Syntax

```
function P1_GetParameterData(NodeAddress : Word;
    Index : Cardinal;
    Item : Word;
    var Buffer;
    BufSize : Cardinal) : Integer;
```

Description

This function retrieves information about a parameter from the diagnostic data. The parameters returned can be any of the following:

PLUS+1® diagnostic base API functions

- OS signals
- Checkpoints
- Set Value signals (writable)
- Parameter Alias (non-volatile, writable)
- Dynamic NV signals (non-volatile, writable)

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
Index	The number of the parameter to retrieve information, beginning at 0 (zero) and ending at the ParameterCount value returned from the function `P1_GetParameterCount` minus 1.
Item*	Selects what type of data to retrieve.
	0 = **Parameter name**
	1 = **Parameter type** (See *PLUS+1® parameter types*).
	2 = **Writable parameter check** (Returns W if the parameter is writable with `P1_WriteNVParameter` or `P1_WriteNVParameterList`, otherwise an empty string).
	3 = **Parameter dimension** (Value in enclosed in brackets defined the array length. Accessible value is from index 0..Length-1. Parameter dimension will only be returned for parameters which contain [x]).
	4 = **Parameter Read** access level (0-9,D)
	5 = **Parameter Write** access level (0-9,D) (Only available for writable parameters).
BufSize	The size of the buffer to which the buffer points.

* All data is returned as null-terminated character strings.

Output parameters

Parameter	Description
Buffer	The address of a buffer which is to receive the data.

PLUS+1® parameter types

Type	Minimum	Maximum
BOOL	0	1
U8	0	255
U16	0	65535
U32	0	4294967295
S8	-128	127
S16	-32768	32767
S32	-2147483648	2147483647

Return value

Value	Description
-1	Success

PLUS+1® diagnostic base API functions

User errors

Error	Description
202	Invalid node address
501	API not started
502	Diagnostic data not loaded
514	Invalid Item value
515	Invalid Index value
516	Buffer to small
529	Write access level not available

P1_ReadParameter

C++ Syntax

```
int P1_ReadParameter(unsigned short NodeAddress,
    char* ParameterName,
    char* Buffer,
    unsigned int BufSize);
```

Delphi Syntax

```
function P1_ReadParameter(NodeAddress : Word;
    ParameterName : PAnsiChar;
    var Buffer;
    BufSize : Cardinal) : Integer;
```

Description

This function reads the value of a parameter from a node.

Only OS signals, Parameter Alias, and Dynamic NV signals are allowed to be read.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
ParameterName	The name of the parameter to read.
BufSize	The size of the buffer to which the buffer points.

Output parameters

Parameter	Description
Buffer	The address of a buffer which is to receive the parameter value. All data is returned as null- terminated character strings.

Return value

Value	Description
-1	Success

PLUS+1® diagnostic base API functions

User errors

Error	Description
138	Unsupported parameter type
201	API function already running
202	Invalid node address
204	No answer from node
221	ParameterName not found
322	Access denied to parameter
501	API not started
502	Diagnostic data not loaded
516	Buffer to small

ECU errors

Error	Description
203	Invalid answer
323	Unable to read parameter

P1_ReadParameterArray

C++ Syntax

```
int P1_ReadParameterArray(unsigned short NodeAddress,
    char* ParameterArrayName,
    char* Range,
    char* Buffer,
    unsigned int BufSize);
```

Delphi Syntax

```
function P1_ReadParameterArray(NodeAddress : Word;
    ParameterArrayName : PAnsiChar;
    Range : PAnsiChar;
    var Buffer;
    BufSize : Cardinal) : Integer;
```

Description

This function reads the value of a parameter from a node.

Only arrays are allowed to be read.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
ParameterArrayName	The name of the parameter to read. The parameter must be an array or array of struct or array of struct with array member etc. In any case, there must be exactly one occurrence of the substring "[]" in the ParameterArrayName. Example: "MyArray[]"

PLUS+1® diagnostic base API functions

Parameter	Description
Ranges	This is a list of comma-separated ranges to log from in the array specified by ParameterArrayName. The substring "[]" in the ParameterName is replaced with the values specified by range. A valid range can be either an integer or two integers with a minus "-" sign between them. The ranges shall be enclosed by hard brackets. Example: "[1-3,6,7-9]" All values must be within the array.
BufSize	The size of the buffer to which the buffer points

The input parameters are checked in the following order:

1. Check that the substring "[]" occurs once in ParameterArrayName
2. Check that the parameter array exists.

   The parameter must exist with [x] in the parameter list. Example: "CAN[x].Baudrate need to exist if CAN[].Baudrate is passed as ParameterArrayName.
3. Check that the range is valid and read access is allowed.

Output parameters

Parameter	Description
Buffer	The address of a buffer which is to receive the parameter values. All data is returned as null- terminated, comma-separated, character strings.

Return value

Value	Description
-1	Success

User errors

Error	Description
138	Unsupported parameter type
201	API function already running
202	Invalid node address
204	No answer from node
221	Parameter not found
322	Access denied to parameter
501	API not started
502	Diagnostic data not loaded
516	Buffer to small
525	Invalid range
531	Invalid parameter format

ECU errors

Error	Description
203	Invalid answer

PLUS+1® diagnostic base API functions

P1_WriteParameter

C++ Syntax

```
int P1_WriteParameter(unsigned short NodeAddress,
    char* ParameterName,
    char* Value);
```

Delphi Syntax

```
function P1_WriteParameter(NodeAddress : Word;
    ParameterName : PAnsiChar;
    Value : PAnsiChar) : Integer;
```

Description

This function writes a parameter in a node.

Only Parameter Alias, Set Value nd Dynamic NV signals are allowed to be written.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
ParameterName	The name of the parameter to write
Value	Value to write

Return value

Value	Description
-1	Success

User errors

Error	Description
138	Unsupported parameter type
151	Value out of range
201	API function already running
202	Invalid node address
204	No answer from node
221	ParameterName not found
322	Access denied to parameter
501	API not started
502	Diagnostic data not loaded

ECU errors

Error	Description
203	Invalid answer
223	Unable to write parameters
245	Verify failed

PLUS+1® diagnostic base API functions

P1_ClearReadParameterList

C++ Syntax

```
int P1_ClearReadParameterList( );
```

Delphi Syntax

```
function P1_ClearReadParameterList : Integer;
```

Description

This function clears the parameter list used by `P1_ReadParameterList`.

Return value

Value	Description
-1	Success

User errors

Error	Description
501	API not started

P1_AddReadParameterList

C++ Syntax

```
int P1_AddReadParameterList(unsigned short NodeAddress,
    char* ParameterName);
```

Delphi Syntax

```
function P1_AddReadParameterList(NodeAddress : Word;
    ParameterName : PAnsiChar) : Integer;
```

Description

This function adds a parameter to the read parameter list used by `P1_ReadParameterList`.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
ParameterName	The name of the parameter to add.

Return value

Value	Description
-1	Success

PLUS+1® diagnostic base API functions

User errors

Error	Description
201	API function already running
202	Invalid node address
221	ParameterName not found
501	API not started
502	Diagnostic data not loaded
524	Parameter already in list

P1_ReadParameterList

C++ Syntax

```
int P1_ReadParameterList(char* Buffer,
    unsigned int BufSize);
```

Delphi Syntax

```
function P1_ReadParameterList(var Buffer;
    BufSize : Cardinal) : Integer;
```

Description

This function reads the parameter list and stores the parameter values in the buffer.

Only OS signals and Parameter Alias are allowed to be read.

Input parameters

Parameter	Description
BufSize	The size of the buffer to which the buffer points

Output parameters

Parameter	Description
Buffer	The address of a buffer which is to receive the value. Parameter values will be passed as a character string with comma after each parameter. If the function is unable to read a parameter it will pass an empty string. Example: 100,,-5, First parameter value is 100, second is unknown because of a error and third value is -5

Return value

Value	Description
-1	Success

User errors

Error	Description
138	Unsupported parameter type
201	API function already running
204	No answer from node

PLUS+1® diagnostic base API functions

Error	Description
218	Parameter list empty 322
322	Access denied to parameter
501	API not started
502	Diagnostic data not loaded
516	Buffer to small

ECU errors

Error	Description
203	Invalid answer
323	Unable to read parameter

P1_ClearWriteParameterList

C++ Syntax

```
int P1_ClearWriteParameterList( );
```

Delphi Syntax

```
function P1_ClearWriteParameterList : Integer;
```

Description

This function clears the parameter list used by `P1_WriteParameterList`.

Return value

Value	Description
-1	Success

User errors

Error	Description
501	API not started

P1_AddWriteParameterList

C++ Syntax

```
int P1_AddWriteParameterList(unsigned short NodeAddress,
    char* ParameterName,
    char* Value);
```

Delphi Syntax

```
function P1_AddWriteParameterList(NodeAddress : Word;
    ParameterName : PAnsiChar;
    Value : PAnsiChar) : Integer;
```

PLUS+1® diagnostic base API functions

Description

This function adds a parameter to the write parameter list used by `P1_WriteParameterList`.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
ParameterName	The name of the parameter to add.
Value	Value to write.

Return value

Value	Description
-1	Success

User errors

Error	Description
201	API function already running
202	Invalid node address
221	ParameterName not found
501	API not started
502	Diagnostic data not loaded
520	Invalid parameter value
524	Parameter already in list

ECU errors

Error	Description
203	Invalid answer
323	Unable to read parameter

P1_WriteParameterList

C++ Syntax

```
int P1_WriteParameterList(char* Buffer,
    unsigned int BufSize);
```

Delphi Syntax

```
function P1_WriteParameterList(var Buffer;
    BufSize : Cardinal) : Integer;
```

Description

This function writes the parameter list.

Only Parameter Alias and Set Value signals are allowed.

PLUS+1® diagnostic base API functions

Input parameters

Parameter	Description
BufSize	The size of the buffer to which the buffer points

Output parameters

Parameter	Description
Buffer	The address of a buffer which is to receive the value. The buffer will only be used if the function returns a no answer or verify failed error. Parameter values will be passed as a character string with comma after each parameter. Parameters that are successfully written will have an empty string and parameters with errors will be indicated with the character F. Example of a verify failed result: ,F, First parameter was okay and the second parameter failed.

Return value

Value	Description
-1	Success

User errors

Error	Description
138	Unsupported parameter type
151	Value out of range
201	API function already running
204	No answer from node
218	Parameter list empty
322	Access denied to parameter
501	API not started
502	Diagnostic data not loaded

ECU errors

Error	Description
203	Invalid answer
223	Unable to write
245	Verify failed

P1_OpenApplication

C++ Syntax

```
typedef TP1_ProgressFunc(unsigned short PercentCompleted);
    int P1_OpenApplication(unsigned short NodeAddress,
    char* FileName,
    TP1_ProgressFunc ProgressProc);
```

PLUS+1® diagnostic base API functions

Delphi Syntax

```
function P1_OpenApplication(NodeAddress : Word;
    FileName : PAnsiChar;
    ProgressProc : TFarProc) : Integer;
```

Description

This function opens a PLUS+1® downloadable application file for download. The file is checksum verified and matched with the supplied node.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
FileName	The name of the file to download with full path.
ProgressProc	Callback function that gives feedback on the progress (0 to 100 percent). A null pointer can optionally be passed in if the callback functionality is undesired.

Return value

Value	Description
-1	Success

User errors

Error	Description
62	File not found
116	File access denied
201	API function already running
202	Invalid node address
216	No node data retrieved with P1_LoadNodeData
231	File does not match this node
247	File not authorized to be downloaded to this node
501	API not started
270	Unsupported file

File errors

Error	Description
55	Corrupted download file
70	Invalid format in file
243	Invalid download file

P1_SetDownloadSettings

C++ Syntax

```
int P1_SetDownloadSettings(unsigned int DownloadSettings);
```

PLUS+1® diagnostic base API functions

Delphi Syntax

```
function P1_SetDownloadSettings(DownloadSettings: Cardinal) : LongInt;
```

Description

This function can be called optionally before the `P1_DownloadApplication` function to configure how parameters should be handled when downloading a PLUS+1® application. The download setting will be used in application downloads until `P1_Close` is called or until `P1_SetDownloadSettings` is called with a different download setting.

Input parameters

Parameter	Description
DownloadSettings	Selects what download settings to use.
	0 = Skip (Default value, parameters will be left unchanged)
	1 = Automatically fix parameters regardless of access level
	2 = Set all parameters to zero, except parameters with read and write access set to Disable (D). Disabled parameters will be left unchanged.
	3 = User Defined. New parameter values for the application to be downloaded is set using `P1_SetDownloadParameterValue`.

P1_GetDownloadData

C++ Syntax

```
int P1_GetDownloadData(unsigned int Item,
    char* Buffer,
    unsigned int BufSize);
```

Delphi Syntax

```
function P1_GetDownloadData(Item : Cardinal;
     var Buffer;
     BufSize : Cardinal) : LongInt;
```

Description

This function retrieves information about the opened download file.

PLUS+1® diagnostic base API functions

Input parameters

Parameter	Description
Item	Selects what type of data to retrieve:
	0 = Filename opened for download
	1 = Application Size number of bytes in hexadecimal notation.
	2 = Highest used Address in hexadecimal
	3 = File OS family
	4 = File OS identity
	5 = File Application Type
	6 = File Application TimeKey
	7 = Node OS family
	8 = Node OS identity
	9 = Node Application Type
	10 = Node Application TimeKey
	11 = LHX Readme File Size
	12 = LHX Readme File Binary Data Note: RTF document encoded as an ANSI string
	13 = File Application Version
	14 = Node Application Version
	15 = File Application ID
	16 = Node Application ID
	17 = (Deprecated)
	18 = (Deprecated)
	19 = ECU application uses dynamic allocation of parameters (1=Yes, 0=No)
	20 = File application uses dynamic allocation of parameters (1=Yes, 0=No)
BufSize	The size of the buffer to which the buffer points.

Output parameters

Parameter	Description
Buffer	The address of a buffer which is to receive the data.

P1_GetDownloadSummaryCount

C++ Syntax

```
int P1_GetDownloadSummaryCount(unsigned int* SummaryCount);
```

Delphi Syntax

```
function P1_GetDownloadSummaryCount(var SummaryCount : Cardinal) :
LongInt;
```

Description

This function returns the total number of parameters that can be read and changed in the currently opened application file. The application remains open until downloaded with `P1_DownloadApplication`.

PLUS+1® diagnostic base API functions

Output parameters

Parameter	Description
SummaryCount	The total number of download summary records.

P1_GetDownloadSummary

C++ Syntax

```
int P1_GetDownloadSummary(unsigned int Index,
    unsigned int Item,
    char* Buffer,
    unsigned int BufSize);
```

Delphi Syntax

```
function P1_GetDownloadSummary(Index,Item : Cardinal;
    var Buffer;
    BufSize : Cardinal) : LongInt;
```

Description

This function retrieves download summary information for the currently opened application.

Input parameters

Parameter	Description	
Index	The index of the download summary record to retrieve information from, a value between 0 and SummaryCount -1.	
Item	Selects what type of data to retrieve.	
	0 = Record type	0 = Parameter type changed
		1 = New parameter
		2 = Deleted parameter
		3 = Common parameter
	1 = Parameter Name	
	2 = Current parameter type	
	3 = New parameter type in application to be downloaded	
	4 = Current ECU value for command parameters and parameters with changed type Value will be updated after calling `P1_SetDownloadParameterValue`	
	5 = ECU value after download, updated after calling `P1_DownloadApplication`	
	6 = Parameter setting result, updated after calling `P1_DownloadApplication`	
	7 = Download parameter (1=Yes, 0=No), updated when calling `P1_SetDownloadParameterValue`	
BufSize	The size of the buffer to which the buffer points.	

Output parameters

Parameter	Description
Buffer	The address of a buffer which is to receive the data.

PLUS+1® diagnostic base API functions

P1_SetDownloadParameterValue

C++ Syntax

```
int P1_SetDownloadParameterValue(char* ParameterName,
char* Value);
```

Delphi Syntax

```
function P1_SetDownloadParameterValue(
    ParameterName,Value : PAnsiChar) : LongInt;
```

Depends on

`P1_OpenApplication`. `P1_GetDownloadSummary` for parameter name

`P1_SetDownloadSettings` (DownloadSettings = 3)

Description

This function sets a parameter value in non-volatile memory for the currently opened application. The application remains open until downloaded with `P1_DownloadApplication`.

Input parameters

Parameter	Description
ParameterName	The name of the parameter to download.
Value	The parameter value to download.

P1_DownloadApplication

C++ Syntax

```
typedef TP1_ProgressFunc(unsigned short PercentCompleted);

int P1_DownloadApplication(unsigned short NodeAddress,
    TP1_ProgressFunc ProgressProc);
```

Delphi Syntax

```
function P1_DownloadApplication(NodeAddress : Word;
    ProgressProc : TFarProc) : Integer;
```

Description

This function downloads a file loaded with the function `P1_OpenApplication` to a node.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
ProgressProc	Callback function that gives feedback on the progress (0 to 100 percent). A null pointer can optionally be passed in if the callback functionality is undesired.

PLUS+1® diagnostic base API functions

Return value

Value	Description
-1	Success

User errors

Error	Description
201	API function already running
202	Invalid node address
204	No answer from node
216	No node data retrieved with `P1_LoadNodeData`
236	No file opened with `P1_OpenApplication`
501	API not started

ECU errors

Error	Description
248	Security access denied
251	Node reset failed
252	Unit history update failed
253	Request of download checksum failed
254	Calculation of download checksum failed
255	Transfer exit failed
256	Erase flash request failed
257	Erase flash failed
258	Download request okay size missing
259	Download request failed
260	Unable to write unit history
261	Unable to start download session
266	Download transfer timeout
267	Download transfer failed
321	Download disabled by node application

P1_DownloadCancel

C++ Syntax

```
int P1_DownloadCancel( );
```

Delphi Syntax

```
function P1_DownloadCancel : Integer;
```

Description

This function cancels an ongoing download started with `P1_DownloadApplication`.

PLUS+1® diagnostic base API functions

Return value

Value	Description
-1	Success

User errors

Error	Description
501	API not started
521	No download started

P1_RecoverECU

C++ Syntax

```
typedef TP1_ProgressFunc(unsigned short PercentCompleted);

int P1_RecoverECU(unsigned short NodeAddress_In,
    unsigned char RecoveryMode,
    unsigned short msTimeout,
    TP1_ProgressFunc ProgressProc,
    unsigned short* NodeAddress_Out);
```

Delphi Syntax

```
function P1_RecoverECU(NodeAddress_In: Word;
    RecoveryMode : Byte;
    msTimeout : Word;
    TP1_ProgressFunc ProgressProc;
    var NodeAddress_Out: Word) : Integer;
```

Description

This function attempts to catch an ECU in boot mode before it enters the application mode, and then keep the ECU in boot mode. This command is very special, because it requires some help from the user. Here are the requirements:

1. The CAN bus must have at least 3 units attached; The diagnostic interface, the node to recover and at least one healthy unit that can set the ack bit in the CAN messages sent by the diagnostic interface.
2. The user must power off the unit to recover, and only that unit, before calling this function.
3. The user must call this function and then power on the unit within the specified timeout.
4. Same DLL and ECU baudrate.

Input parameters

Parameter	Description
NodeAddress_In	The address of the node to send the request.
RecoveryMode	The method of recovery. The following values are supported:
	0 = Recover the node with the specified NodeAddress_In
	1 = Recover the first PLUS+1® node found on the bus. NodeAddress_In is ignored.
msTimeout	The timeout in milliseconds.
ProgressProc	Callback function that gives feedback on the progress (0 to 100 percent). A null pointer can optionally be passed in if the callback functionality is undesired.

PLUS+1® diagnostic base API functions

Output parameters

Parameter	Description
NodeAddress_Out	The address of the node recovered (if any).

Return value

Value	Description
-1	Success

User errors

Error	Description
530	Invalid ECU recovery mode

P1_Silent

C++ Syntax

```
int P1_Silent(unsigned short NodeAddress,
    unsigned char Options);
```

Delphi Syntax

```
function P1_Silent(NodeAddress : Word;
    Options : Byte): LongInt;
```

Description

This function sets single or all nodes in “Silent” mode. Nodes in “Silent” mode will ignore any command from the PLUS+1® Diagnostic API except for: `P1_Silent`

Input parameters

Parameter	Description
NodeAddress *	The address to send the request.
Options	The silent mode of the nodes. The following values are supported:
	0 = Normal all nodes (not silent)
	1 = Silent all nodes
	2 = Normal (not silent)
	3 = Silent

* Only used when Option is 2 or 3.

Return value

Value	Description
-1	Success

PLUS+1® diagnostic base API functions

User errors

Error	Description
520	Invalid parameter value

ECU errors

Error	Description
202	Invalid node address
203	Invalid answer
204	No answer from node
248	Security access denied

Read only parameter API functions

P1_GetReadOnlyParameterFiles

C++ Syntax

```
int P1_GetReadOnlyParameterFiles(unsigned short NodeAddress,
    unsigned short* FileCount);
```

Delphi Syntax

```
function P1_GetReadOnlyParameterFiles(NodeAddress : Word;
    var FileCount : Word) : Integer;
```

Description

This function returns the total number of read only parameter files for a node.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.

Output parameters

Parameter	Description
FileCount	A pointer to a DWORD which will receive the total number of read only parameter files.

Return value

Value	Description
-1	Success

User errors

Error	Description
202	Invalid node address
501	API not started
502	Diagnostic data not loaded

P1_GetReadOnlyParameterFileData

C++ Syntax

```
int P1_GetReadOnlyParameterFileData(unsigned short NodeAddress,
    unsigned short FileIndex,
    unsigned short Item,
    char* Buffer,
    unsigned int BufSize);
```

Read only parameter API functions

Delphi Syntax

```
function P1_GetReadOnlyParameterFileData(NodeAddress : Word;
    FileIndex : Word;
    Item : Word;
    var Buffer;
    BufSize : Cardinal) : Integer;
```

Description

This function retrieves information about a read only parameter file. It is necessary to run this function repeatedly with different Item values to read out a all information for a read only parameter file.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
FileIndex	The index of the read only parameter file to retrieve information, beginning at 0 (zero) and ending at the FileCount value returned from the function `P1_GetReadOnlyParameterFiles` minus 1.
Item	Selects what type of Read only parameter data to retrieve.
	0 = **File type**
	1 = **File status** (returns 1 if file is valid otherwise 0)
	2 = **File name**
	3 = **Version**
	4 = **TimeKey**
	5 = **File size** (in decimal format)
	All data is returned as null-terminated character strings.
BufSize	The size of the buffer to which the buffer points.

Output parameters

Parameter	Description
Buffer	The address of a buffer which is to receive the data.

Return value

Value	Description
-1	Success

User errors

Error	Description
202	Invalid node address
211	Unsupported infoblock
501	API not started
502	Diagnostic data not loaded
514	Invalid Item value
515	Invalid Index value
516	Buffer to small

Read only parameter API functions

P1_OpenReadOnlyParameterFile

C++ Syntax

```
typedef TP1_ProgressFunc(unsigned short PercentCompleted);

int P1_OpenReadOnlyParameterFile(unsigned short NodeAddress,
    unsigned short FileIndex,
    char* FileName,
    TP1_ProgressFunc ProgressProc);
```

Delphi Syntax

```
function P1_OpenReadOnlyParameterFile(NodeAddress : Word;
    FileIndex : Word;
    FileName : PAnsiChar;
    ProgressProc : TFarProc) : Integer;
```

Description

This function opens a PLUS+1® downloadable read only parameter file for download. The file is checksum verified and matched with the supplied node.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
FileIndex	The index of the read only parameter file to retrieve information, beginning at 0 (zero) and ending at the FileCount value returned from the function `P1_GetReadOnlyParameterFiles` minus 1.
FileName	The name of the file to download with full path.
ProgressProc	Callback function that gives feedback on the progress (0 to 100 percent). A null pointer can optionally be passed in if the callback functionality is undesired.

Return value

Value	Description
-1	Success

User errors

Error	Description
62	File not found
116	File access denied
201	API function already running
202	Invalid node address
216	No node data retrieved with P1_LoadNodeData
231	File does not match this node
239	Invalid file index
247	File not authorized to be downloaded to this node
501	API not started
270	Unsupported file
527	No read only parameter file information available

Read only parameter API functions

File errors

Error	Description
70	Invalid format in file
243	Invalid download file

P1_DownloadReadOnlyParameterFile

C++ Syntax

```
typedef TP1_ProgressFunc(unsigned short PercentCompleted);
    int P1_DownloadReadOnlyParameterFile(unsigned short NodeAddress,
    TP1_ProgressFunc ProgressProc);
```

Delphi Syntax

```
function P1_DownloadReadOnlyParameterFile(NodeAddress : Word;
    ProgressProc : TFarProc) : Integer;
```

Description

This function downloads a file loaded with the function `P1_OpenReadOnlyParameterFile` to a node.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
ProgressProc	Callback function that gives feedback on the progress (0 to 100 percent). A null pointer can optionally be passed in if the callback functionality is undesired.

Return value

Value	Description
-1	Success

User errors

Error	Description
201	API function already running
202	Invalid node address
204	No answer from node
216	No node data retrieved with `P1_LoadNodeData`
236	No file opened with `P1_OpenReadOnlyParameterFile`
501	API not started

ECU errors

Error	Description
248	Security access denied
251	Node reset failed

Read only parameter API functions

Error	Description
252	Unit history update failed
253	Request of download checksum failed
254	Calculation of download checksum failed
255	Transfer exit failed
256	Erase flash request failed
257	Erase flash failed
258	Download request okay size missing
259	Download request failed
260	Unable to write unit history
261	Unable to start download session
266	Download transfer timeout
267	Download transfer failed

Application log API functions

P1_GetApplicationLogFiles

C++ Syntax

```
int P1_GetApplicationLogFiles(unsigned short NodeAddress,
    unsigned short* FileCount);
```

Delphi Syntax

```
function P1_GetApplicationLogFiles(NodeAddress : Word;
    var FileCount : Word) : Integer;
```

Description

This function retrieves the total number of application log files for a node.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.

Output parameters

Parameter	Description
FileCount	A pointer to a DWORD which will receive total number of application log files.

Return value

Value	Description
-1	Success

User errors

Error	Description
201	API function already running
202	Invalid node address
501	API not started
502	Diagnostic data not loaded

P1_GetApplicationLogFileData

C++ Syntax

```
int P1_GetApplicationLogFileData(unsigned short NodeAddress,
    unsigned short FileIndex,
    unsigned short Item,
    char* Buffer,
    unsigned int BufSize);
```

Application log API functions

Delphi Syntax

```
function P1_GetApplicationLogFileData(NodeAddress : Word;
    FileIndex : Word;
    Item : Word;
    var Buffer;
    BufSize : Cardinal) : Integer;
```

Description

This function returns one record of the application log files defined in the application.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
FileIndex	The index number of the application log file data record to retrieve information, beginning at 0 (zero) and ending at the RecordCount value returned from the function `P1_GetApplicationLogFiles` minus 1.
Item	Selects what type of data to retrieve.
	0 = **FileName**
	All data is returned as null-terminated character strings.
BufSize	

Output parameters

Parameter	Description
Buffer	The address of a buffer which is to receive the data.

Return value

Value	Description
-1	Success

User errors

Error	Description
202	Invalid node address
501	API not started
502	Diagnostic data not loaded
514	Invalid Item value
515	Invalid Index value
516	Buffer to small

P1_ReadApplicationLog

C++ Syntax

```
typedef TP1_ProgressFunc(unsigned short PercentCompleted);
int P1_ReadApplicationLog(unsigned short NodeAddress,
    unsigned short FileIndex,
```

Application log API functions

```
    TP1_ProgressFunc ProgressProc,
    unsigned int* RecordCount);
```

Delphi Syntax

```
function P1_ReadApplicationLog(NodeAddress : Word;
    FileIndex : Word;
    ProgressProc : TFarProc;
    var RecordCount : Cardinal) : Integer;
```

Description

This function reads specified application log file from a node. The application log information can be read out with `P1_GetReadApplicationLogData`.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
FileIndex	The index number of the application log file to read, beginning at 0 (zero) and ending at the FileIndex value returned from the function `P1_GetApplicationLogFiles` minus 1.
ProgressProc	Callback function that gives feedback on the progress (0 to 100 percent). A null pointer can optionally be passed in if the callback functionality is undesired.

Output parameters

Parameter	Description
RecordCount	A pointer to a LWORD which will receive total number of application log records.

Return value

Value	Description
-1	Success

User errors

Error	Description
201	API function already running
204	No answer from node
501	API not started
502	Diagnostic data not loaded
526	Access denied to function

ECU errors

Error	Description
203	Invalid answer
248	Security access denied

Application log API functions

P1_GetApplicationLogData

C++ Syntax

```
int P1_GetApplicationLogData(unsigned short NodeAddress,
    unsigned short FileIndex,
    unsigned int RecordIndex,
    unsigned short Item,
    char* Buffer,
    unsigned int BufSize);
```

Delphi Syntax

```
function P1_GetApplicationLogData(NodeAddress : Word;
    FileIndex : Word;
    RecordIndex : Cardinal;
    Item : Word;
    var Buffer;
    BufSize : Cardinal) : Integer;
```

Description

This function returns one record of the application log. It is necessary to run this function repeatedly with different Item values to read out a complete application log record.

Input parameters

Parameter	Description
NodeAddress	The address of the node to send the request. The most significant byte is the net number and the least significant byte is the node number.
FileIndex	The index number of the application log file to retrieve information, beginning at 0 (zero) and ending at the FileCount value returned from the function `P1_GetApplicationLogFiles` minus 1.
RecordIndex	The index number of the application log record to retrieve information, beginning at 0 (zero) and ending at the RecordCount value returned from the function `P1_ReadApplicationLog` minus 1.
Item *	Selects what type of data to retrieve.
	0 = **Tag**
	E = Error
	S = Statistic
	O = Other
	1 = **Access**
	2 = **Data**
BufSize	The size of the buffer to which the buffer points.

* All data is returned as null-terminated character strings.

Output parameters

Parameter	Description
Buffer	The address of a buffer which is to receive the data.

Application log API functions

Return value

Value	Description
-1	Success

User errors

Error	Description
501	API not started
502	Diagnostic data not loaded
514	Invalid Item value
515	Invalid Index value
516	Buffer to small
526	Access denied to function

Return code explanation

Return codes

Return codes explanation

Code	Type	Description	Explanation
-1	Success	Function returned without any error or warnings.	
62	User	File not found.	Supplied file could not be found. Verify that a full path and extension is supplied to the file.
70	File	Invalid format in file.	File may be generated with a new software version that is unsupported.
116	User	File access denied.	Read access is required for the file. Check file security.
138	User	Unsupported parameter type.	Selected parameter name is not supported by the API. For valid parameter types see *P1_GetParameterData* on page 21.
151	User	Value out of range.	For parameter value ranges see *P1_GetParameterData* on page 21
161	User	Invalid API license key.	The API license key or DLL key supplied in function `P1_Open` is not valid.
166	ECU	Unknown key value.	ECU application is invalid and needs to be updated to allow diagnostics.
200	File	Error loading gateway drivers.	Check that the required DLL files are available.
201	User	API function already running.	Call to a function has not finished. Wait until previous call finishes.
202	User	Invalid node address.	Supplied node address is not available in the system.
203	ECU	Invalid answer.	ECU application answered with an invalid response.
204	User	No answer from node.	Check cabling and make sure that ECU is powered.
205	User	Invalid baudrate.	Selected baudrate is not supported. For a list of valid baudrates see *P1_Open* on page 10.
211	ECU	Unsupported infoblock.	Unsupported ECU application.
212	ECU	Unable to load diagnostic data.	Error occurred while loading the diagnostic data.
213	ECU	Unable to restore diagnostic data.	Error occurred while decoding the diagnostic data.
214	ECU	Unable to read diagnostic data.	Error occurred while uploading the diagnostic data from the ECU.
215	Warning	No valid diagnostic data available.	ECU application does not contain any diagnostic data. May be caused by ECU running in boot mode. ECU will start in boot mode if `P1_DownloadCancel` is used. File Download is still possible.
216	User	No node data retrieved with `P1_LoadNodeData`.	`P1_LoadNodeData` must be successfully executed for the ECU.
218	User	Parameter list empty.	No parameters added with `P1_AddReadParameterList` or `P1_AddWriteParameterList`.

Return code explanation

Return codes explanation (continued)

Code	Type	Description	Explanation
221	User	ParameterName not found.	Supplied parameter name was not found in the diagnostic data. Use `P1_GetParameterData` to verify that the parameter name is valid.
223	ECU	Unable to write parameters.	Error occurred while writing the parameters to the ECU.
231	User	File does not match this node.	Download file does not match the ECU. Ensure that the correct NodeAddress is used and read ECU application information with `P1_GetNodeInfo`.
236	User	No file opened with `P1_OpenApplication`.	
238	ECU	Infoblock checksum error.	ECU application is invalid and needs to be updated to allow diagnostics.
241	ECU	Unable to read node data.	ECU application is invalid and needs to be updated to allow diagnostics.
242	User	Incorrect toolkey.	Access restricted to ECU because the supplied toolkey is incorrect.
243	License	Invalid LicenseKey.	Supplied LicenseKey is invalid.
243	File	Invalid download file.	Supplied download file is invalid.
245	ECU	Verify failed.	Write parameter failed because the verify value was different from the written value. May indicate a faulty ECU or may occur if the ECU application sets the parameter value.
247	User	File not authorized to be downloaded to this node.	File does not match ECU part number or serial number.
248	ECU	Security access denied.	Error occurred while communicating with the ECU.
249	ECU	Unable to start diagnostic session.	Error occurred while communicating with the ECU.
250	ECU	Invalid diagnostic data length.	ECU answered with a invalid diagnostic data length.
251	ECU	Node reset failed.	File download failed because of incorrect answer from ECU.
252	ECU	Unit history update failed.	File download failed because of incorrect answer from ECU.
253	ECU	Request of download checksum failed.	File download failed because of incorrect answer from ECU .
254	ECU	Calculation of download checksum failed.	File download failed because of incorrect answer from ECU.
255	ECU	Transfer exit failed.	File download failed because of incorrect answer from ECU.
256	ECU	Erase flash request failed.	File download failed because of incorrect answer from ECU.
257	ECU	Erase flash failed.	File download failed because of incorrect answer from ECU.
258	ECU	Download request okay size missing.	File download failed because of incorrect answer from ECU.

Return code explanation

Return codes explanation (continued)

Code	Type	Description	Explanation
259	ECU	Download request failed.	File download failed because of incorrect answer from ECU.
260	ECU	Unable to write unit history.	File download failed because of incorrect answer from ECU.
261	ECU	Unable to start download session.	File download failed because of incorrect answer from ECU.
263	User	Invalid application download file.	This error may be returned if an Read Only Parameter download file is opened with `P1_OpenApplication`.
265	User	Security access timeout.	Last call to `P1_LoadNodeData` failed because of incorrect toolkey. Wait 10 seconds before calling `P1_LoadNodeData` with a different toolkey.
266	ECU	Download transfer timeout.	File download failed because of incorrect answer from ECU.
267	ECU Download transfer failed	File download failed because of incorrect answer from ECU.	
268	Warning	Access denied to application without toolkey.	Toolkey is required for ECU applications developed in GUIDE. File download is still possible.
270	User	Unsupported file.	
271	ECU	Diagnostic Data Error: Index out of range.	
272	ECU	Diagnostic Data Error: Invalid address vector length.	
273	ECU	Diagnostic Data Error: No end tag.	
274	ECU	Diagnostic Data Error: Signal outside struct group.	
275	ECU	Diagnostic Data Error: Invalid data type.	
276	ECU	Diagnostic Data Error: Invalid flag.	
277	ECU	Diagnostic Data Error: Invalid struct format.	
278	ECU	Diagnostic Data Error: Invalid struct group format.	
279	ECU	Diagnostic Data Error: Invalid array range.	
280	ECU	Diagnostic Data Error: Invalid end of data.	
316	User	Unable to read application log.	Make sure that a system scan is performed after an ECU has been power cycled before reading application log data.
319	User	Application log file empty.	
321	User	Download disabled by node application.	
322	User	Access denied to parameter.	

Return code explanation

Return codes explanation (continued)

Code	Type	Description	Explanation
323	ECU	Unable to read parameter.	
500	User	API already started.	Run `P1_Close` before running `P1_Open`.
501	User	API not started.	`P1_Open` must be successfully executed.
502	User	Diagnostic data not loaded.	`P1_LoadNodeData` must be successfully executed for the ECU.
503	License	Incorrect LicenseKey version.	Unsupported LicenseKey.
504	User	Incorrect API DLL filename.	API DLL filename needs to be Plus1diag.dll
507	User	Invalid gateway selection.	See *P1_Open* on page 10 for a list of valid gateways.
508	User	Gateway drivers not found.	Drivers for the selected gateway were not found. Check that the drivers are properly installed.
509	User	DiagnosticFilePath not found.	Check that a full path is supplied.
510	User	License does not support Customer ID.	
511	User	Invalid Customer ID value.	See *P1_SetCustomerID* on page 12 for valid Customer ID ranges.
512	User	No nodes found.	Check cabling and make sure that ECU is powered.
513	User	Invalid node index.	
514	User	Invalid item value.	
515	User	Invalid index value.	
516	User	Buffer to small.	Buffer is too small to contain the result string.
517	ECU	Unable to read node type.	Unsupported ECU application.
518	User	Access to node denied.	For supported ECU applications, see *ECU application support* on page 5.
519	ECU	Unable to read unit history.	Error occurred while communicating with the ECU.
520	User	Invalid parameter value.	Supplied parameter value is not a valid number value.
521	User	No download started.	A file download must be started with `P1_DownloadApplication` before using this function.
522	User	Gateway Error.	Unable to connect to selected gateway. Check driver installation and make sure it is connected.
523	User	LicenseKey has expired.	
524	User	Parameter already in list.	Same parameter name can not be added to the list more than once.
525	User	Invalid range.	
526	ECU	Access denied to function.	The ECU application has denied access to the function.
527	User	No read only parameter file information available.	
528	User	Parameter access denied.	Access restricted by ECU application.
529	User	Write access level not available.	Write access level is only available for writable parameters.

Return code explanation

Return codes explanation (continued)

Code	Type	Description	Explanation
530	User	Invalid ECU recovery mode.	
531	User	Invalid parameter format.	
533	User	Invalid option value.	Invalid option value in `P1_Open` options parameter.
534	ECU	No data available.	
536	User	File does not match target.	The selected download file does not match the target ECU.
537	Warning	Parameter setup changed.	
538	ECU	Unable to find ECU after file download.	This error can be returned when a ECU application with a different baud rate is downloaded.
539	ECU	CRC not calculated.	The file download could not be verified because an error during the CRC calculation.
540	ECU	CRC calculation failed.	The file download could not be verified because an error during the CRC calculation.
541	ECU	CRC algorithm failed.	The file download could not be verified because an error during the CRC calculation.
542	ECU	CRC calculation ongoing.	The file download could not be verified because an error during the CRC calculation.
543	ECU	CRC value missing.	The file download could not be verified because an error during the CRC calculation.
544	ECU	Unable to verify CRC.	The file download could not be verified because an error during the CRC calculation.
545	ECU	No Application Log data available.	
546	User	Read only parameter data structure not matching.	The Read Only Parameter file does not match the selected ECU.
547	User	Read only parameter type not matching.	The Read Only Parameter file does not match the selected ECU.
548	User	Invalid read only parameter checksum.	The Read Only Parameter file does not match the selected ECU.
549	User	Read only parameter size not matching.	The Read Only Parameter file does not match the selected ECU.
550	User	Invalid option name.	
553	Warning	No Diagnostic Data file installed.	
554	Warning	Diagnostic Data file already installed.	
555	User	File does not contain a Diagnostic Data file.	
556	User	DLL function missing in protocol DLL.	
557	User	Invalid LHX file.	
558	ECU	ECU is in boot loader mode.	Download a valid ECU application.
1003	User	RP1210 DLL not found.	
1004	User	CG150 DLL not found.	
1500	ECU	Function not supported by ECU.	

RP1210A requirements for PLUS+1® diagnostics

RP1210A functions used

RP1210_ClientDisconnect hwndClient

Parameter	Parameter value
hwndClient	Always 0
nDeviceId	Taken from the ini file
fpchhProtocol	CAN
	Format 4 now
	Format 1, 2, 3 soon
lTxBufferSize	Always 0 (default 8k)
lRcvBufferSize	Always 0 (default 8k)
nIsAppPacketizingIncomingMsgs	Always 0

RP1210_ClientDisconnect

Parameter	Parameter value
nClientID	Returned value from ClientConnect

RP1210_SendMessage

Parameter	Parameter value
nClientID	Returned value from ClientConnect
fpchClientMessage	Pointer to buffer
nMessageSize	Length of CAN data + 5 n
NotifyStatusOnTx	Always 0
nBlockOnSend	Preferably 1, 0 only if necessary

RP1210_ReadMessage

Parameter	Parameter value
nClientID	Returned value from ClientConnect
fpchAPIMessage	Pointer to 18 byte char array
nBufferSize	Always 18
nBlockOnRead	Preferably 1, 0 only if necessary

RP1210_SendCommand (called up to 3 times in sequence)

Parameter		Parameter value
1 (Always):	nCommandNumber	3
	nClientID	Returned value from ClientConnect
	fpchClientCommand	0
	nMessageSize	0
2 (Optional):	nCommandNumber	5
	nClientID	Returned value from ClientConnect
	fpchClientCommand	{1,0x0F,0xFF,0,0,0x0C,0xDA,0,0}
	nMessageSize	9

RP1210A requirements for PLUS+1® diagnostics

Parameter		Parameter value
3 (Optional):	nCommandNumber	5
	nClientID	Returned value from ClientConnect
	fpchClientCommand	{1,0x0F,0xFF,0,0,0x0C,0xDB,0,0}
	nMessageSize	9

RP1210_ReadVersion

Parameter	Parameter value
fpchDLLMajorVersion	Pointer to 2 byte char array
fpchDLLMinorVersion	Pointer to 2 byte char array
fpchAPIMajorVersion	Pointer to 2 byte char array
fpchAPIMinorVersion	Pointer to 2 byte char array RP1210A

RP1210_GetErrorMsg

Not used at present.

RP1210_GetHardwareStatus

Not used at present.

RP1210_ReadDetailedVersion

Parameter	Parameter value
fpchAPIVersionInfo	Pointer to 17 byte char array
fpchDLLVersionInfo	Pointer to 17 byte char array
fpchFWVersionInfo	Pointer to 17 byte char array

Function usage in depth

RP1210_ClientConnect

- Called at init
- Required to exist in the DLL

RP1210_ClientDisconnect

- Called at exit
- Expected to interrupt any blocking calls made to other functions in the DLL
- Required to exist in the DLL

RP1210_SendCommand

- Called once, immediately after `RP1210_ClientConnect` to set all filters to pass
- Optionally, it can be called 2 more times to allow CAN IDs:
  - CDAxxxx
  - CDBxxxx
  - Required to exist in the DLL

RP1210_SendMessage

- Called in its own thread

RP1210A requirements for PLUS+1® diagnostics

- Receives messages to send from the main thread via a pipe
- If possible a blocking call will be made
- Otherwise, and if ERR_TX_QUEUE_FULL is received, it will wait for some time and then retry
- Message formatting is always for CAN:
  - "Message type" 1 byte, always = 1
  - "CAN ID" always 4 bytes
    BIG ENDIAN default
    Can also handle little endian
  - "Data" 0-8 bytes
- Required to exist in the DLL

RP1210_ReadMessage

- Called in its own thread
- Sends received messages to the main thread via a pipe
- If possible a blocking call will be made
- Otherwise, it will poll
- Message formatting is always for CAN:
  - "Time stamp" 4 bytes, BIG ENDIAN
    Timestamp is not used for anything at the moment
  - "ECHO" always 0 bytes– –
  - "Message type" 1 byte.
  - "CAN ID" always 4 bytes
    BIG ENDIAN default
    Can also handle little endian
  - "Data" 0-8 bytes
- Required to exist in the DLL

RP1210_ReadDetailedVersion

- Called once at init
- Not required to exist in the DLL

RP1210_ReadVersion

- Called once at init if `RP1210_ReadDetailedVersion` does not exist in the DLL
- Not required to exist in the DLL

RP1210_GetErrorMsg

- Not used
- Not required to exist in the DLL

RP1210_GetHardwareStatus

- Not used
- Not required to exist in the DLL

Products we offer:

- Bent Axis Motors
- Closed Circuit Axial Piston Pumps and Motors
- Displays
- Electrohydraulic Power Steering
- Electrohydraulics
- Hydraulic Power Steering
- Integrated Systems
- Joysticks and Control Handles
- Microcontrollers and Software
- Open Circuit Axial Piston Pumps
- Orbital Motors
- PLUS+1® GUIDE
- Proportional Valves
- Sensors
- Steering
- Transit Mixer Drives

Danfoss Power Solutions is a global manufacturer and supplier of high-quality hydraulic and electronic components. We specialize in providing state-of-the-art technology and solutions that excel in the harsh operating conditions of the mobile off-highway market. Building on our extensive applications expertise, we work closely with our customers to ensure exceptional performance for a broad range of off-highway vehicles.

We help OEMs around the world speed up system development, reduce costs and bring vehicles to market faster.

Danfoss – Your Strongest Partner in Mobile Hydraulics.

Go to www.powersolutions.danfoss.com for further product information.

Wherever off-highway vehicles are at work, so is Danfoss. We offer expert worldwide support for our customers, ensuring the best possible solutions for outstanding performance. And with an extensive network of Global Service Partners, we also provide comprehensive global service for all of our components.

Please contact the Danfoss Power Solution representative nearest you.

Comatrol
www.comatrol.com

Schwarzmüller-Inverter
www.schwarzmueller-inverter.com

Turolla
www.turollaocg.com

Hydro-Gear
www.hydro-gear.com

Daikin-Sauer-Danfoss
www.daikin-sauer-danfoss.com

Local address:

Danfoss
Power Solutions (US) Company
2800 East 13th Street
Ames, IA 50010, USA
Phone: +1 515 239 6000

Danfoss
Power Solutions GmbH & Co. OHG
Krokamp 35
D-24539 Neumünster, Germany
Phone: +49 4321 871 0

Danfoss
Power Solutions ApS
Nordborgvej 81
DK-6430 Nordborg, Denmark
Phone: +45 7488 2222

Danfoss
Power Solutions Trading (Shanghai) Co., Ltd.
Building #22, No. 1000 Jin Hai Rd
Jin Qiao, Pudong New District
Shanghai, China 201206
Phone: +86 21 3418 5200