ASAM GDI DCDParser

API Documentation

rd electronic GmbH

location Zwickau

Newtonstraße 12a

08060 Zwickau

Tel. 0375-447990-70

Internet: http://www.rd-electronic.de

Document Number:	DCDParser-APIDoc
Issue:	1.1.2.2
Status:	Release
Last Modification:	2021-03-30
Created on:	2017-12-04
Authors:	KSc (rde) SBa (rde)

Important Notice

Document Change History

Version	Date	Changed by	Modification
1.1.0.0	2017-12-04	SBa	Created Original “GDI DCDParser43 UserMan”
1.1.0.1	2017-12-20	SBa	Refine, corrections,
1.1.1.0	2017-12-21	SBa	Return types changed from size_t to unsigned long
1.1.1.1	2017-12-22	SBa	Correction CInterface – non support of DIT content Correction of technical terms
1.1.2.0	2018-01-04	SBa/KSc	Refinement
1.1.2.1	2018-02-05	SBa/KSc	Refinement (GetTextByDITAccess, GetTextByResult)
1.1.2.2	2021-03-30	SBa	Update company information

Document Approval Sheet

Document Number:	DCDParser-APIDoc
Title:	ASAM GDI DCDParser
Subtitle:	API Documentation
Issue:	1.1.2.2
Date:	2021-03-30
Status:	Release

Contents

1 General

1.1 Identification

This document describes the C++ -API of the ASAM GDI DCDParser of rd electronic GmbH.

The DCDParser API allows the user of the DCDParser to get information about the contained objects of the parsed DCD.

The DCDParser is compatible to the standards DCD 4.2 up to DCD 4.4.

2 Generell methods

This chapter lists methods that are implemented within different classes of the DCDParser API.

2.1 Iteration methods for Data Types

2.1.1 GetFirstDataType

Declaration:

```
bool GetFirstDataType (const CDataType *&rpoDataType)
```

Parameter `rpoDataType`:

Output parameter for the first Data Type.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

2.1.2 `GetNextDataType`

There are two possibilities to use this method.

- Use to previous Data Type pointer as iterator to get the next pointer (parameter 2).

  This way is save for multithreaded actions.

- Use the internal iterator to get the next Data Type pointer (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextDataType(
    const CDataType *&rpoNextDataType,
    const CDataType *poPreviousDataType = NULL)
```

Parameter `rpoNextDataType`:

Outout parameter for the next Data Type.

Parameter `poPreviousDataType`:

Input parameter of the previous Data Type.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

2.1.3 GetDataTypeByName

Declaration:

```
bool GetDataTypeByName(
    const CDataType *&rpoDataType,
    const char* szDataTypeName )
```

Parameter `rpoDataType`:

Output parameter for the Data Type with the specified name.

Parameter `szDataTypeName`:

Input parameter for the name to search for.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

2.2 Iteration methods for Constants

2.2.1 GetFirstConstant

Declaration:

```
bool GetFirstConstant (const CConstant *&rpoConstant)
```

Parameter `rpoConstant`:

Output parameter for the first constant.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

2.2.2 `GetNextConstant`

There are two possibilities to use this method.

- Use to previous constant pointer as iterator to get the next pointer (parameter 2).

  This way is save for multithreaded actions.

- Use the internal iterator to get the next constant pointer (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextConstant(
    const CConstant *&rpoNextConstant,
    const CConstant *poPreviousConstant = NULL)
```

Parameter `rpoNextConstant`:

Outout parameter for the next constant.

Parameter `poPreviousConstant`:

Input parameter of the previous constant.

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

2.2.3 GetConstantByName

Declaration:

```
bool GetConstantByName(
    const CConstant *&rpoConstant,
    const char* szConstName )
```

Parameter rpoConstant:

Output parameter for the constant with the specified name

Parameter szConstName:

Input parameter for the name to search for

Return:

true	Output parameter is initialized
false	Output parameter is not set

2.3 Iteration methods for Modules

2.3.1 GetFirstModule

Declaration:

```
bool GetFirstModule (const CModule *&rpoModule)
```

Parameter rpoModule:

Output parameter for the first Module

Return:

true	Output parameter is initialized
false	Output parameter is not set

2.3.2 GetNextModule

There are two possibilities to use this method.

- Use to previous Module pointer as iterator to get the next pointer (parameter 2).

  This way is save for multithreaded actions.

- Use the internal iterator to get the next Module pointer (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextModule(
    const CModule *&rpoNextModule,
    const CModule *poPreviousModule = NULL )
```

Parameter `rpoNextModule`:

Outout parameter for the next Module.

Parameter `poPreviousModule`:

Input parameter of the previous Module.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

2.3.3 GetModuleByID

Declaration:

```
bool GetModuleByID(
    const CModule *&rpoModule,
    unsigned long ulModuleID )
```

Parameter `rpoModule`:

Output parameter for the Module with the specified ID.

Parameter `ulModuleID`:

Input parameter for the id to search for.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

2.3.4 GetModuleByName

Declaration:

```
bool GetModuleByName(
    const CModule *&rpoModule,
    const char* szModuleName )
```

Parameter `rpoModule`:

Output parameter for the Module with the specified name.

Parameter `szModuleName`:

Input parameter for the name to search for

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

2.4 Iteration methods for Interfaces

2.4.1 GetFirstInterface

Declaration:

```
bool GetFirstInterface (const CInterface *&rpoInterface)
```

Parameter `rpoInterface`:

Output parameter for the first interface

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

2.4.2 `GetNextInterface`

There are two possibilities to use this method.

- Use to previous interface pointer as iterator to get the next pointer (parameter 2).

  This way is save for multithreaded actions.

- Use the internal iterator to get the next interface pointer (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextInterface(
    const CInterface *&rpoNextInterface,
    const CInterface * poPreviousInterface = NULL )
```

Parameter `rpoNextInterface`:

Outout parameter for the next interface

Parameter `poPreviousInterface`:

Input parameter of the previous interface

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

2.4.3 `GetInterfaceByID`

Declaration:

```
bool GetInterfaceByID(
    const CInterface *&rpoInterface,
    unsigned long ulInterfaceID )
```

Parameter `rpoInterface`:

Output parameter for the interface with the specified ID

Parameter `ulInterfaceID`:

Input parameter for the id to search for

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

2.4.4 `GetInterfaceByName`

Declaration:

```
bool GetInterfaceByName(
    const CInterface *&rpoInterface,
    const char* szInterfaceName )
```

Parameter `rpoInterface`:

Output parameter for the interface with the specified name.

Parameter `szInterfaceName`:

Input parameter for the name to search for.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

2.5 Iteration methods for Operations

This chapter lists methods to retrieve all Operations of a Class.

2.5.1 `GetFirstOperation`

Declaration:

```
bool GetFirstOperation (const COperation *&rpoOperation)
```

Parameter `rpoOperation`:

Output parameter for the first Operation.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

2.5.2 GetNextOperation

There are two possibilities to use this method.

- Use to previous Operation pointer as iterator to get the next pointer (parameter 2).

  This way is save for multithreaded actions.

- Use the internal iterator to get the next Operation pointer (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextOperation(
    const COperation *&rpoNextOperation,
    const COperation *poPreviousOperation = NULL )
```

Parameter `rpoNextOperation`:

Outout parameter for the next Operation .

Parameter `poPreviousOperation`:

Input parameter of the previous Operation.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

2.5.3 `GetOperationByID`

Declaration:

```
bool GetOperationByID(
    const COperation *&rpoOperation,
    unsigned long ulOperationID )
```

Parameter `rpoOperation`:

Output parameter for the Operation with the specified ID.

Parameter `ulOperationID`:

Input parameter for the id to search for.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

2.5.4 GetOperationByName

Declaration:

```
bool GetOperationByName(
    const COperation *&rpoOperation,
    const char* szOperationName )
```

Parameter `rpoOperation`:

Output parameter for the Operation with the specified name.

Parameter `szOperationName`:

Input parameter for the name to search for

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

2.6 Methods to access content of DITs

2.6.1 GetInfo

Supported by DCD of version 4.2.

Fill a DIT Information object of `struct CDITAccess` [3.13].

Declaration:

```
bool GetInfo(
    const DCDParser::CDITAccess *&rpoDITAccess ) const;
```

2.6.2 GetError

Supported by DCD of version 4.2.

Fill a DIT Error object of `struct CDITAccess` [3.13].

Declaration:

```
bool GetError(
    const DCDParser::CDITAccess *&rpoDITAccess ) const;
```

2.6.3 GetQuestion

Supported by DCD of version 4.2.

Fill a DIT Question object of `struct CDITAccess` [3.13].

Declaration:

```
bool GetQuestion(
    const DCDParser::CDITAccess *&rpoDITAccess ) const;
```

2.6.4 GetMessage

Supported by DCD of version 4.3.
Fill a DIT Message object of `struct CDITAccess` [3.13].

Declaration:

```
bool GetMessage(
    const DCDParser::CDITAccess *&rpoDITAccess ) const;
```

2.6.5 GetText

Supported by DCD of version 4.3.

Fill a DIT Text object of `struct CDITAccess` [3.13].

Declaration:

```
bool GetText(
    const DCDParser::CDITAccess *&rpoDITAccess ) const;
```

2.7 Methods of essential type objects

2.7.1 GetMin

Get the minimum value of Type object.

Declaration:

```
char             GetMin()
unsigned char    GetMin()
short            GetMin()
unsigned short   GetMin()
long             GetMin()
unsigned long    GetMin()
float            GetMin()
double           GetMin()
```

Return:

Minimum of current data type defined within DCD.

2.7.2 GetMax

Get the maximum value of Type object.

Declaration:

```
<T> GetMax()
```

Return:

Maximum of current data type defined within DCD.

2.7.3 GetStep

Get the increment between sequenced values of Type object.

Declaration:

```
<T> GetStep()
```

Return:

Step size of current data type defined within DCD.

2.8 Methods for declaration information

2.8.1 GetDeclarationFile

Returns the full path of file, within the DCD object is declared.

Declaration:

```
const char*   GetDeclarationFile() const;
```

Return:

File path of declaration file.

2.8.2 `GetDeclarationLine`

Returns the line number in declaration file.

Declaration:

```
unsigned long GetDeclarationLine() const;
```

Return:

Line number in declaration file.

3 DCDParser API

To start the DCDParser it is necessary to create an instance of the `class CDCD` [3.1]. The method `Parse` [3.1.6] of this starts the parse process.

The return value of the method is a pointer to an object of `class CParseStatus` [3.2]. This object contains information about the parse process.

3.1 class CDCD

The class is the main class to start the parse process and to get parsed content of DCDs.

3.1.1 Iteration methods for Data Types

The class supports the set of Iteration methods for Data Types [2.1].

3.1.2 Iteration methods for Constants

The class supports the set of Iteration methods for Constants [2.2].

3.1.3 Iteration methods for Modules

The class supports the set of Iteration methods for Modules [2.3].

3.1.4 Iteration methods for Interfaces

The class supports the set of Iteration methods for Interfaces [2.4].

3.1.5 Iteration methods for Operations

The class supports the set of Iteration methods for Operations [2.5].

3.1.6 `Parse`

After the creation of the instance of the `class CDCD` [3.1], it is necessary to call the method “Parse”. This method parses the specified DCD with all included files.

Declaration:

```
const CParseStatus* Parse(
    char* szFileToParse,
    bool bDCDVersionCheckOn = true,
    unsigned char* SequenceAllocateFunc(
                unsigned long ulSize) = NULL,
    short SequenceFreeFunc(
                unsigned char* pucData) = NULL,
    unsigned char* SequenceReallocFunc (
                unsigned char* pucData,
                unsigned long ulSize) = NULL )
```

Parameter `szFileToParse`:

DCD file to parse

Parameter `bDCDVersionCheckOn`:

Flag for DCDVersionCheck (Default = `true`)

Parameter `SequenceAllocateFunc`:

Pointer to the function `os_allocate` (Default = `NULL`)

Do not use if `os_allocate` is not available.

Parameter `SequenceFreeFunc`:

Pointer to the function `os_free` (Default = `NULL`)

Do not use if `os_free` is not availbale.

Parameter `SequenceReallocFunc`:

Pointer to the function `os_reallocate` (Default = `NULL`)

Do not use if `os_reallocate` is not available.

Return:

Pointer to object with information about the parse process.

Example:

```
// SequenceAllocateFunc not used
// SequenceFreeFunc not used
// SequenceReallocFunc not used

int main(int argc, char* argv[])
{
    CDCD oDCD;
    const CParseStatus* poParseStatus;
    poParseStatus = oDCD.Parse(argv[1]);
        …
    return 0;
}
```

3.1.7 `Parse`

After the creation of the instance of the `class CDCD` [3.1], it is necessary to call the method “Parse”. This method parses the specified DCD with all included files.

Declaration:

```
const DCDParser::CParseStatus* Parse (
    const char* szFileToParse,
    unsigned short unAlignment,
    bool bDCDVersionCheckOn = true,
    unsigned char* DCD43_PA_CALL SequenceAllocateFunc(
                unsigned long ulSize) = NULL,
    short DCD43_PA_CALL SequenceFreeFunc(
                unsigned char* pucData ) = NULL,
    unsigned char* DCD43_PA_CALL SequenceReallocFunc (
                unsigned char* pucData,
                unsigned long ulSize ) = NULL )
```

Parameter `szFileToParse`:

DCD file to parse

Parameter `unAlignment`:

Alignment of Device Driver.

Parameter `bDCDVersionCheckOn`:

Flag for DCDVersionCheck (Default = `true`)

Parameter `SequenceAllocateFunc`:

Pointer to the function `os_allocate` (Default = `NULL`)

Do not use if `os_allocate` is not available.

Parameter `SequenceFreeFunc`:

Pointer to the function `os_free` (Default = `NULL`)

Do not use if `os_free` is not availbale.

Parameter `SequenceReallocFunc`:

Pointer to the function `os_reallocate` (Default = `NULL`)

Do not use if `os_reallocate` is not available.

Return:

Pointer to object with information about the parse process.

3.1.8 GetParserVersion

This method returns the version string of the DCDParser.

Declaration:

```
const char* GetParserVersion()
```

3.1.9 GetDCDVersion

This method returns the version string of the parsed DCD.

Declaration:

```
TeDCDVersion GetDCDVersion()
```

Return:

```
DCD_VERSION_42              // DCD version 4.2
DCD_VERSION_43              // DCD version 4.3
DCD_VERSION_44              // DCD version 4.4
DCD_VERSION_45              // DCD version 4.5
DCD_VERSION_UNDEFINED       // version check turned off
```

3.1.10 GetDeclarationFile

Returns the full path of file, within the DCD object is declared.

Declaration:

```
const char*   GetDeclarationFile() const;
```

Return:

File path of declaration file.

3.1.11 `GetDeclarationLine`

Returns the line number in declaration file.

Declaration:

```
unsigned long GetDeclarationLine() const;
```

Return:

Line number in declaration file.

3.1.12 `GetDIT`

This method returns an object to access to the parsed DIT.

Declaration:

```
bool   GetDIT(CDIT &roDIT)
```

Parameter `roDIT`:

Output parameter for the DIT object of `class CDIT` [3.3].

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

3.1.13 `GetNumberOfTypes`

Returns the number of global DCD Types.

Declaration:

```
unsigned long GetNumberOfTypes()     const;
```

3.1.14 GetNumberOfConstants

Returns the number of global DCD Constants.

Declaration:

```
unsigned long GetNumberOfConstants() const;
```

3.1.15 GetNumberOfOperations

Returns the number of global DCD Operations.

Declaration:

```
unsigned long GetNumberOfOperations()const;
```

3.1.16 GetNumberOfInterfaces

Returns the number of global DCD Interfaces.

Declaration:

```
unsigned long GetNumberOfInterfaces()const;
```

3.1.17 GetNumberOfModules

Returns the number of global DCD Modules.

Declaration:

```
unsigned long GetNumberOfModules()   const;
```

3.1.18 GetNumberOfParsedFiles

Returns the number of all parsed DCD and header files.

Declaration:

```
unsigned long GetNumberOfParsedFiles() const;
```

3.1.19 GetParsedFileByIndex

Returns the file path of parsed file with the index from 0 to `GetNumberOfParsedFiles` [3.1.18].

Declaration:

```
const char* GetParsedFileByIndex(
    unsigned long ulParsedFileIndex) const;
```

3.1.20 GetVersionGDI

Returns the GDI version from the DCD Header information.

Declaration:

```
unsigned long GetVersionGDI()    const;
```

3.1.21 GetVersionDriver

Returns the version of the Device Driver from the DCD Header information.

Declaration:

```
unsigned long GetVersionDriver() const;
```

3.1.22 GetVersionDevice

Returns the version of the Device from the DCD Header information.

Declaration:

```
unsigned long GetVersionDevice() const;
```

3.1.23 GetVersionDCD

Returns the DCD version from the DCD Header information.

Declaration:

```
unsigned long GetVersionDCD()    const;
```

3.1.24 GetDriverName

Returns the name of the Device Driver from the DCD Header information.

Declaration:

```
const char* GetDriverName()    const;
```

3.1.25 GetFactory

Returns the factory of the Device Driver from the DCD Header information.

Declaration:

```
const char* GetFactory()       const;
```

3.1.26 GetNumberOfDITs

Returns the number of DIT files from the DCD Header information.

Declaration:

```
unsigned long GetNumberOfDITs() const;
```

3.1.27 GetDITNameByIndex

Returns the file name of DIT file entry with the index from 0 to `GetNumberOfDITs` [3.1.26].

Declaration:

```
const char* GetDITNameByIndex(
    unsigned long ulDITIndex) const;
```

3.1.28 GetDITLanguageByIndex

Returns the language assigned to a DIT file entry with the index from 0 to `GetNumberOfDITs` [3.1.26].

Declaration:

```
const char* GetDITLanguageByIndex(
    unsigned long ulDITIndex) const;
```

3.1.29 GetNumberOfCompanions

Return the number of Companions entries from the DCD Header information.

Declaration:

```
unsigned long GetNumberOfCompanions() const;
```

3.1.30 `GetCcompanionByIndex`

Returns the Companion entry from DCD Header information with the index from 0 to GetNumberOfCompanions [3.1.29].

Declaration:

```
const char* GetCompanionByIndex(
    unsigned long ulCompanionIndex) const;
```

3.2 `class CParseStatus`

The class CParseStatus includes information of the parse process. The class contains the return code, the parse message, the last parsed file and the last parsed line. This class provides also the possibility to get the version of the DCDParser.

The DCDParser API classes should only be used if the return code is PARSE_OK or WARNING. If warnings occurred during the parse process then it is necessary to check if these warnings affect the following usage.

3.2.1 GetReturnCode

Get the return code of the parse process `Parse` [3.1.6].

Declaration:

```
TeErrorCode GetReturnCode()
```

Return:

`PARSE_OK`	parse process runs without errors
`WARNING`	parse process runs with warnings
`SYNTACTICAL_ERROR`	syntactical error
`SEMANTICAL_ERROR`	semantically error
`INTERNAL_ERROR`	error in the DCD43Parser
`INPUT_ERROR`	DCD or include file not found
`DCD_VERSION_ERROR`	used rules do not fit to DCD version
`ERROR_CODE_NOT_SET`	error detected by the parser but the error status is not set

3.2.2 GetMessage

Get the message of the parse process `Parse` [3.1.6].

Declaration:

```
const char* GetMessage()
```

Return:

Message text.

3.2.3 GetErrorMessage

Get the error message of the parse process `Parse` [3.1.6].

Declaration:

```
const char* GetErrorMessage()
```

Return:

Error message text.

3.2.4 GetErrorFile

Get file name that contains the parse error.

Declaration:

```
const char* GetErrorFile()
```

Return:

File name.

3.2.5 GetErrorLine

Get the line number of error within file that contains the parse error.

Declaration:

```
unsigned long GetErrorLine()
```

Return:

Line number of error.

3.2.6 GetNumberOfWarnings

This method returns the number of occurred warnings during the parse process.

Declaration:

```
unsigned long GetNumberOfWarnings()
```

Return:

Number of warnings.

3.2.7 GetWarningMessageByNumber

This method returns the message of the specified warning by the number of the warning.

Declaration:

```
const char* GetWarningMessageByNumber()
```

Return:

Message text.

3.2.8 GetWarningFileByNumber

This method returns the name of the file where the specified warning occurred.

Declaration:

```
const char* GetWarningFileByNumber()
```

Return:

File name.

3.2.9 GetWarningLineByNumber

This method returns the line in the file where the specified warning occurred.

Declaration:

```
unsigned long GetWarningLineByNumber()
```

Return:

Line within file.

3.3 class CDIT

This class is able to access DIT files which are specified within the Header information of the parsed DCD. To parse a certain DIT file it is necessary to call the method `SetLanguage` [3.3.2]. You have to call this method before using over methods of this class except the methods `SetComErrorDIT` [3.3.1] and `GetParserVersion` [3.3.5].

3.3.1 SetComErrorDIT

Use this method to specify the path to the Com Error DIT file. This method does not parse the standard DIT file but it checks if the path to the file is valid. If the specified path is valid, then the Com Error DIT file will be parse by calling the method `SetLanguage` [3.3.2].

Declaration:

```
bool SetComErrorDIT(const char* szComErrorDIT)
```

Parameter `szComErrorDIT`:

Path to Com Error DIT file.

Return:

`true`	File was found.
`false`	File was not found.

3.3.2 `SetLanguage`

This method searches in the Header information of the parsed DCD if the specified DIT language is available. If the language is valid, then the related DIT is parsed. The method returns the parse status of the DIT parse process. This method has to be called before using over methods of the `class CDIT` [3.3].

If the Com Error DIT file is specified via method `SetComErrorDIT` [3.3.1], then the Com Error DIT file is also parsed by this method.

Declaration:

```
CParseStatus* SetLanguage (const char* szDITLanguage)
```

Parameter `szDITLanguage`:

Language of the DIT file.

Return:

Object with information of the parse process.

3.3.3 GetTextByDITAccess

Returns the text of DIT entry identified by given section numbers from the DIT file.

Declaration:

```
const char* GetTextByDITAccess(
    CDITAccess* poDITAccess,
    bool* pbIDValid = NULL)
```

Parameter value `poDITAccess`:

DIT entry of `struct CDITAccess` [3.13].

Parameter `pbIDValid`:

Output parameter with the status of all other output parameters after call of method.

`true`	Valid.
`false`	Invalid.

Return:

DIT file text message that belongs to the DIT entry.

3.3.4 GetTextByResult

Returns the full text of DIT entry returned by Device Driver.

Declaration:

```
const char* GetTextByResult(
    CResult*          poResult,
    short             nReturnValue,
    char*             pcAllocMemory,
    unsigned long     ulMemorySize,
    bool*             pbValid = NULL );
```

Parameter `poResult`:

Input pointer to result structure:

```
typedef struct
{
    short m_nQual;        // qual
    short m_nGrade;       // grade
    short m_nCode;        // code
    char* m_pszzInsTxt;   // double zero terminated string
} CResult;
```

Parameter `nReturnValue`:

Return code.

Parameter `pcAllocMemory`:

Pointer to allocated memory for the text.

Paremeter `ulMemorySize`:

Size of the allocated memory.

Parameter `pbValid`:

Output parameter with the status of all other output parameters after call of method.

`true`	Output parameters are initialized.
`false`	Output parameters are not set.

Return:

DIT file text message that belongs to the DIT Result.

3.3.5 GetParserVersion

This method returns the version string of the used DITParser.

Declaration:

```
const char* GetParserVersion()
```

3.3.6 GetDeclarationFile

Returns the full path of DIT file.

Declaration:

```
const char* GetDeclarationFile() const;
```

Return:

Path of declaration file.

3.3.7 GetNumberOfParsedFiles

Returns the number of all parsed *.dit and *.dii files.

Declaration:

```
unsigned long GetNumberOfParsedFiles() const;
```

3.3.8 GetParsedFileByIndex

Returns the file path of parsed file with the index from 0 to `GetNumberOfParsedFiles` [3.3.7].

Declaration:

```
const char* GetParsedFileByIndex(
    unsigned long ulParsedFileIndex ) const;
```

3.4 class CModule

The class represents an Module was declared within parsed DCD.

3.4.1 Iteration methods for Constants

The class supports the set of Iteration methods for Constants [2.2].

3.4.2 Iteration methods for Interfaces

The class supports the set of Iteration methods for Interfaces [2.4].

3.4.3 Iteration methods for Operations

The class supports the set of Iteration methods for Operations [2.5].

3.4.4 Methods to access content of DITs

The class supports the set of Methods to access content of DITs [2.6].

3.4.5 Methods for declaration information

The class supports the set of Methods for declaration information [2.8].

3.4.6 GetDeclarationModule

Returns the exterior Module in which this Module was declared.

Declaration:

```
bool GetDeclarationModule(
    const DCDParser::CModule *&rpoModule ) const;
```

Parameter rpoModule:

DCD Module of `class CModule` [3.4].

3.4.7 GetID

The method returns the id of the current Module.

Declaration:

```
unsigned long GetID()
```

Return:

Id of the current Module.

3.4.8 GetName

The method returns the name of the current Module.

Declaration:

```
const char*   GetName()
```

Return:

Name of the current Module.

3.4.9 `GetFirstInitiateParam`

Declaration:

```
bool GetFirstInitiateParam (
    const DCDParser::CInitiateParam *&rpoInitiateParam)
```

Parameter `rpoInitiateParam`:

Output parameter for the first Initiate Parameter

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

3.4.10 `GetNextInitiateParam`

There are two possibilities to use this method.

- Use to previous constant pointer as iterator to get the next pointer (parameter 2).

  This way is save for multithreaded actions.

- Use the internal iterator to get the next constant pointer (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextInitiateParam(
    const CInitiateParam *&rpoNextInitiateParam,
    const CInitiateParam *poPreviousInitiateParam = NULL)
```

Parameter `rpoNextInitiateParam`:

Outout parameter for the next Initiate Parameter.

Parameter `poPreviousInitiateParam`:

Input parameter of the previous Initiate Parameter.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.4.11 `GetInitiateParamByName`

Declaration:

```
bool GetInitiateParamByName(
    const CInitiateParam *&rpoInitiateParam,
    const char* szInitiateParamName )
```

Parameter `rpoInitiateParam`:

Output parameter for the Initiate Parameter with the specified name.

Parameter `szInitiateParamName`:

Input parameter for the name to search for.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.4.12 `GetInitiateParamStruct`

The method returns all Initiate Parameters of the current Module. If there is only one Initiate Parameter, then the method returns a reference to this Initiate Parameter. If there is more than one Initiate Parameter, then the method returns a structure of all Initiate Parameter.

Declaration:

```
bool GetInitiateParamStruct(
    const CInitiateParam *&rpoInitiateParam)
```

Parameter value:

`rpoInitiateParam` Output parameter for all Initiate Parameters.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.4.13 `IsControlVD`

The Module is a Control VD.

Declaration:

```
bool IsControlVD(void) const
```

3.4.14 `IsControlVD`

The Module identified by given ID is a Control VD.

Declaration:

```
static bool IsControlVD(unsigned long ulID)
```

3.5 class CInterface

The class represents an Interface was declared within parsed DCD.

3.5.1 Iteration methods for Operations

The class supports the set of Iteration methods for Operations [2.5].

3.5.2 Methods for declaration information

The class supports the set of Methods for declaration information [2.8].

3.5.3 GetDeclarationModule

Declaration:

```
bool GetDeclarationModule(
    const DCDParser::CModule *&rpoModule ) const;
```

3.5.4 GetID

The method returns the id of the current interface.

Declaration:

```
unsigned long GetID()
```

Return:

Id of the current interface.

3.5.5 GetName

The method returns the name of the current interface.

Declaration:

```
const char*   GetName()
```

Return:

Name of the current interface.

3.5.6 IsAbstract

Declaration:

```
bool  IsAbstract()
```

Return:

`true`	Interface is abstract.
`false`	Interface is not abstract.

3.5.7 GetFirstOperationChildOnly

Declaration:

```
bool GetFirstOperationChildOnly (
    const COperation *&rpoOperation)
```

Parameter `rpoOperation`:

Output parameter for the first Operation.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.5.8 `GetNextOperationChildOnly`

There are two possibilities to use this method.

- Use to previous Operation pointer as iterator to get the next pointer (parameter 2).

  This way is save for multithreaded actions.

- Use the internal iterator to get the next Operation pointer (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextOperationChildOnly(
    const COperation *&rpoNextOperation,
    const COperation *poPreviousOperation = NULL )
```

Parameter `rpoNextOperation`:

Output parameter for the next Operation

Parameter `poPreviousOperation`:

Input parameter of the previous Operation

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

3.5.9 `GetOperationByIDChildOnly`

Declaration:

```
bool GetOperationByIDChildOnly(
    const COperation *&rpoOperation,
    unsigned long ulOperationID )
```

Parameter `rpoOperation`:

Output parameter for the Operation with the specified ID.

Parameter `ulOperationID`:

Input parameter for the id to search for.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.5.10 GetOperationByNameChildOnly

Declaration:

```
bool GetOperationByNameChildOnly (
    const COperation *&rpoOperation,
    const char* szOperationName )
```

Parameter `rpoOperation`:

Output parameter for the Operation with the specified name.

Parameter `szOperationName`:

Input parameter for the name to search for

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

3.5.11 GetFirstCreateParam

Declaration:

```
bool GetFirstCreateParam (
    const DCDParser::CCreateParam *&rpoCreateParam)
```

Parameter `rpoCreateParam`:

Output parameter for the first Create Parameter

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

3.5.12 `GetNextCreateParam`

There are two possibilities to use this method.

- Use to previous constant pointer as iterator to get the next pointer (parameter 2).

  This way is save for multithreaded actions.

- Use the internal iterator to get the next constant pointer (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextCreateParam(
    const CCreateParam *&rpoNextCreateParam,
    const CCreateParam *poPreviousCreateParam = NULL)
```

Parameter `rpoNextCreateParam`:

Outout parameter for the next Create Parameter.

Parameter `poPreviousCreateParam`:

Input parameter of the previous Create Parameter.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.5.13 `GetCreateParamByName`

Declaration:

```
bool GetCreateParamByName(
    const CCreateParam *&rpoCreateParam,
    const char* szCreateParamName )
```

Parameter `rpoCreateParam`:

Output parameter for the Create Parameter with the specified name.

Parameter `szCreateParamName`:

Input parameter for the name to search for.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.5.14 `GetCreateParamStruct`

The method returns all Create Parameters of the current Module. If there is only one Create Parameter, then the method returns a reference to this Create Parameter. If there is more than one Create Parameter, then the method returns a structure of all Create Parameters.

Declaration:

```
bool GetCreateParamStruct(
    const CCreateParam *&rpoCreateParam)
```

Parameter value:

`rpoCreateParam`	Output parameter for all Create Parameters.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.5.15 `GetFirstCreateParamChildOnly`

Declaration:

```
bool GetFirstCreateParamChildOnly (
    const DCDParser::CCreateParam *&rpoCreateParam)
```

Parameter `rpoCreateParam`:

Output parameter for the first Create Parameter

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

3.5.16 `GetNextCreateParamChildOnly`

There are two possibilities to use this method.

- Use to previous constant pointer as iterator to get the next pointer (parameter 2).

  This way is save for multithreaded actions.

- Use the internal iterator to get the next constant pointer (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextCreateParamChildOnly (
    const CCreateParam *&rpoNextCreateParam,
    const CCreateParam *poPreviousCreateParam = NULL)
```

Parameter `rpoNextCreateParam`:

Outout parameter for the next Create Parameter.

Parameter `poPreviousCreateParam`:

Input parameter of the previous Create Parameter.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.5.17 `GetCreateParamByNameChildOnly`

Declaration:

```
bool GetCreateParamByNameChildOnly (
    const CCreateParam *&rpoCreateParam,
    const char* szCreateParamName )
```

Parameter `rpoCreateParam`:

Output parameter for the Create Parameter with the specified name.

Parameter `szCreateParamName`:

Input parameter for the name to search for.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.5.18 `GetCreateParamStructChildOnly`

The method returns all Create Parameters of the current Module. If there is only one Create Parameter, then the method returns a reference to this Create Parameter. If there is more than one Create Parameter, then the method returns a structure of all Create Parameters.

Declaration:

```
bool GetCreateParamStruct(
    const CCreateParam *&rpoCreateParam)
```

Parameter value:

`rpoCreateParam`	Output parameter for all Create Parameters.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.5.19 `GetFirstAttParam`

Declaration:

```
bool GetFirstAttParam (
    const DCDParser::CAttParam *&rpoAttParam)
```

Parameter `rpoAttParam`:

Output parameter for the first Attribute.

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

3.5.20 `GetNextAttParam`

There are two possibilities to use this method.

- Use to previous constant pointer as iterator to get the next pointer (parameter 2).

  This way is save for multithreaded actions.

- Use the internal iterator to get the next constant pointer (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextAttParam(
    const CAttParam *&rpoNextAttParam,
    const CAttParam *poPreviousAttParam = NULL)
```

Parameter `rpoNextAttParam`:

Outout parameter for the next Attribute.

Parameter `poPreviousAttParam`:

Input parameter of the previous Attribute.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.5.21 GetAttParamByID

Declaration:

```
bool GetAttParamByID(
    const CAttParam *&rpoAttParam,
    unsigned long ulAttParamID )
```

Parameter `rpoAttParam`:

Output parameter for the Attribute with the specified ID.

Parameter `ulAttParamID`:

Input parameter for the id to search for.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.5.22 GetAttParamByName

Declaration:

```
bool GetAttParamByName(
    const CAttParam *&rpoAttParam,
    const char* szAttParamName )
```

Parameter `rpoAttParam`:

Output parameter for the Attribute with the specified name.

Parameter `szAttParamName`:

Input parameter for the name to search for.

3.5.23 `GetFirstAttParamChildOnly`

Declaration:

```
bool GetFirstAttParamChildOnly (
    const DCDParser::CAttParam *&rpoAttParam)
```

Parameter `rpoAttParam`:

Output parameter for the first Attribute.

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

3.5.24 `GetNextAttParamChildOnly`

There are two possibilities to use this method.

- Use to previous constant pointer as iterator to get the next pointer (parameter 2).

  This way is save for multithreaded actions.

- Use the internal iterator to get the next constant pointer (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextAttParamChildOnly (
    const CAttParam *&rpoNextAttParam,
    const CAttParam *poPreviousAttParam = NULL)
```

Parameter `rpoNextAttParam`:

Outout parameter for the next Attribute.

Parameter `poPreviousAttParam`:

Input parameter of the previous Attribute.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.5.25 GetAttParamByIDChildOnly

Declaration:

```
bool GetAttParamByIDChildOnly (
    const CAttParam *&rpoAttParam,
    unsigned long ulAttParamID )
```

Parameter `rpoAttParam`:

Output parameter for the Attribute with the specified ID.

Parameter `ulAttParamID`:

Input parameter for the id to search for.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.5.26 GetAttParamByNameChildOnly

Declaration:

```
bool GetAttParamByNameChildOnly (
    const CAttParam *&rpoAttParam,
    const char* szAttParamName )
```

Parameter `rpoAttParam`:

Output parameter for the Attribute with the specified name.

Parameter `szAttParamName`:

Input parameter for the name to search for.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.5.27 GetFirstParent

Declaration:

```
bool GetFirstParent   (const CInterface *&proInterface )
```

Parameter `proInterface`:

Output parameter for the first parent interface of the current interface

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.5.28 `GetNextParent`

There are two possibilities to use this method.

- Use to previous Module pointer as iterator to get the next pointer (parameter 2).

  This way is save for multithreaded actions.

- Use the internal iterator to get the next Module pointer (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextParent  (
    const CInterface *&proNextParentInterface,
    const CInterface *poPreviousParentInterface = NULL  )
```

Parameter `proNextParentInterface`:

Output parameter for the next parent interface of the current interface.

Parameter `poPreviousParentInterface`:

Input parameter of the previous parent interface of the current interface.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.6 `class COperation`

The class represents an Operation was declared within parsed DCD.

3.6.1 `Methods to access content of DITs`

The class supports the set of Methods to access content of DITs [2.6].

3.6.2 Methods for declaration information

The class supports the set of Methods for declaration information [2.8].

3.6.3 GetDeclarationInterface

Declaration:

```
Bool GetDeclarationInterface(
    const DCDParser::CInterface *&rpoInterface ) const;
```

3.6.4 GetID

The method returns the id of the current Operation.

Declaration:

```
unsigned long GetID()
```

Return:

Id of the current Operation.

3.6.5 GetName

The method GetName() returns the name of the current Operation.

Declaration:

```
const char*   GetName()
```

Return:

Name of the current Operation.

3.6.6 IsVirtual

Declaration:

```
bool  IsVirtual()
```

Return:

true	Operation is virtual
false	Operation is not virtual

3.6.7 GetFirstInParam

Declaration:

```
bool GetFirstInParam (
    const DCDParser::COperationParam *&rpoInParam)
```

Parameter rpoInParam:

Output parameter for the first Input Parameter.

Return:

true	Output parameter is initialized.
false	Output parameter is not set.

3.6.8 `GetNextInParam`

There are two possibilities to use this method.

- Use to previous constant pointer as iterator to get the next pointer (parameter 2).

  This way is save for multithreaded actions.

- Use the internal iterator to get the next constant pointer (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextInParam (
    const COperationParam *&rpoNextInParam,
    const COperationParam *poPreviousInParam = NULL)
```

Parameter `rpoNextInParam`:

Outout parameter for the next Input Parameter.

Parameter `poPreviousInParam`:

Input parameter of the previous Input Parameter.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.6.9 GetInParamByName

Declaration:

```
bool GetInParamByName (
    const COperationParam *&rpoInParam,
    const char* szInParamName )
```

Parameter `rpoInParam`:

Output parameter for the Input Parameter with the specified name.

Parameter `szInParamName`:

Input parameter for the name of the Input Parameter to search for.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.6.10 GetInParamStruct

The method returns all Input Parameters of the current Operation. If the there is only one Input Parameter, then the method returns a reference to this Input Parameter. If there is more than one Input Parameter, then the method returns a structure of all Input Parameters.

Declaration:

```
bool GetInParamStruct  (const COperationParam *&rpoInParam  )
```

Parameter `rpoInParam`:

Output parameter for all Input Parameters.

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

3.6.11 `GetFirstOutParam`

Declaration:

```
bool GetFirstOutParam (
    const DCDParser::COperationParam *&rpoOutParam)
```

Parameter `rpo`InParam:

Output parameter for the first Output Parameter.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.6.12 `GetNextOutParam`

There are two possibilities to use this method.

- Use to previous constant pointer as iterator to get the next pointer (parameter 2).

  This way is save for multithreaded actions.

- Use the internal iterator to get the next constant pointer (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextOutParam (
    const COperationParam *&rpoNextOutParam,
    const COperationParam *poPreviousOutParam = NULL)
```

Parameter `rpoNextOutParam`:

Outout parameter for the next Output Parameter.

Parameter `poPreviousOutParam`:

Input parameter of the previous Output Parameter.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.6.13 `GetOutParamByName`

Declaration:

```
bool GetOutParamByName (
    const COperationParam *&rpoOutParam,
    const char* szOutParamName )
```

Parameter `rpoOutParam`:

Output parameter for the Output Parameter with the specified name.

Parameter `szOutParamName`:

Input parameter for the name of the Output Parameter to search for.

Return:

`true`	Output parameter is initialized.
`false`	Output parameter is not set.

3.6.14 GetOutParamStruct

The method returns all Output Parameters of the current Operation. If the there is only one Output Parameter, then the method returns a reference to this Output Parameter. If there is more than one Output Parameter, then the method returns a structure of all Output Parameters.

Declaration:

```
bool GetOutParamStruct  (const COperationParam *&rpoOutParam  )
```

Parameter `rpoOutParam`:

Output parameter for all Output Parameters.

Return:

`true`	Output parameter is initialized
`false`	Output parameter is not set

3.7 class CInstance

The class is a base of all DCD Parameters was declared within parsed DCD.

3.7.1 Methods for declaration information

The class supports the set of Methods for declaration information [2.8].

3.7.2 `GetName`

The method returns the name of the current instance.

Declaration:

```
const char*   GetName()
```

Return:

Name of the current instance.

3.7.3 `IsSet`

The method shows if the instance is initialized. If the method returns true, then the memory for the data pointer in the instance is allocated.

Declaration:

```
bool  IsSet()
```

Return:

`true`	instance is initialized
`false`	instance is not initialized

3.7.4 GetDataType

Declaration:

```
bool  GetDataType (const CDataType *&rpoDataType )
```

Parameter `rpoDataType`:

Output parameter for data type of the instance.

Return:

`true`	Data type is set to output parameter.
`false`	Data type of instance is not set.

3.7.5 GetValue

The pre-condition to use this method is that IsSet [3.7.3] returned with `true`.

Declaration:

```
void GetValue( void* pDestination )
```

Parameter `pDestination`:

Output parameter for the data of the instance. This output parameter is a pointer allocated memory.
The size that has to be allocated you get by a call of GetSize [3.14.5] of class CDataType [3.14].
To get the data type call the method GetDataType [3.7.4].

3.7.6 GetDataPtr

This method returns a pointer to the data of the current instance.

Declaration:

```
void* GetDataPtr()
```

Return:

Pointer to the data of the current instance.
`NULL` if the instance is not initialized.

3.8 class CConstant

The class describes the defined constants of the DCD.

3.8.1 Methods of base class CInterface

The class is inherited of class CInstance [3.7] and supports all methods of this base class.

3.9 class CInitiateParam

The class describes the Initiate Parameters of Modules.

3.9.1 Methods of base class CInterface

The class is inherited of class CInstance [3.7] and supports all methods of this base class.

3.9.2 Methods to access content of DITs

The class supports the set of Methods to access content of DITs [2.6].

3.10 class CCreateParam

The class describes the Create Parameters of Interfaces.

3.10.1 Methods of base class CInterface

The class is inherited of class CInstance [3.7] and supports all methods of this base class.

3.10.2 Methods to access content of DITs

The class supports the set of Methods to access content of DITs [2.6].

3.11 class CAttParam

The class describes the Attributes and Parameters of Interfaces.

3.11.1 Methods of base class CInterface

The class is inherited of class CInstance [3.7] and supports all methods of this base class.

3.11.2 Methods to access content of DITs

The class supports the set of Methods to access content of DITs [2.6].

3.11.3 `GetID`

This method returns the id of the parameter or attribute.

Declaration:

```
unsigned long GetID()
```

Return:

Id of the current Parameter or Attribute.

3.11.4 `IsParam`

Declaration:

```
bool IsParam()
```

Return:

`true`	parameter
`false`	attribute

3.11.5 `IsReadOnly`

Declaration:

```
bool IsReadOnly()
```

Return:

`true`	Read only.
`false`	Not read only.

3.12 class COperationParam

The class describes the Input and Output Parameters of Operations.

3.12.1 Methods of base class CInterface

The class is inherited of class CInstance [3.7] and supports all methods of this base class.

3.12.2 Methods to access content of DITs

The class supports the set of Methods to access content of DITs [2.6].

3.13 struct CDITAccess

The structure represents the DIT content of different DCD types.

Declaration:

```
typedef struct
{
    unsigned short m_nDITArea;
    unsigned short m_nDITSubArea;
    unsigned short m_nDITTxtNr;
}CDITAccess;
```

3.14 class CDataType

CDataType is the base class for all data types.

3.14.1 Methods to access content of DITs

The class supports the set of Methods to access content of DITs [2.6].

3.14.2 Methods for declaration information

The class supports the set of Methods for declaration information [2.8].

3.14.3 GetTypeIdentifier

Declaration:

```
const char* GetTypeIdentifier()
```

Return:

Name of the data type.

3.14.4 `GetTypeID`

Declaration:

```
TeDataTypeID GetTypeID()
```

Return:

Id of the current data type:

```
typedef enum
{
    idFLOAT,              // DataTypeFloat
    idDOUBLE,             // DataTypeDouble
    idCHAR,               // DataTypeChar
    idBOOLEAN,        // DataTypeBoolean
    idLONG,               // DataTypeLong
    idSHORT,              // DataTypeShort
    idUNSIGNED_LONG,      // DataTypeUnsignedLong
    idUNSIGNED_SHORT,     // DataTypeUnsignedShort
    idOCTET,              // DataTypeOctet
    idSTRUCT,             // DataTypeStruct
    idENUM,               // DataTypeEnum
    idSEQUENCE,       // DataTypeEnum
    idUNION,              // DataTypeSequence
    idARRAY,              // DataTypeArray
    idINTERFACEREF            // DataTypeInterfaceRef
}TeDataTypeID;
```

3.14.5 `GetSize`

Declaration:

```
unsigned long GetSize()
```

Return:

Size of the current data type in byte.

3.15 class CDataTypeBoolean

The class represents the type `boolean`, was used within parsed DCD.

3.15.1 Methods of base class CDataType

The class supports the methods of class CDataType [3.14].

3.16 class CDataTypeChar

The class represents the type `char`, was used within parsed DCD.

3.16.1 Methods of base class CDataType

The class is inherited of class CDataType [3.14] and supports all methods of this base class.

3.16.2 Methods of essential type objects

The class supports the set of Methods of essential type objects [2.7] for the type `char`.

3.17 class CDataTypeOctet

The class represents the type `unsigned char`, was used within parsed DCD.

3.17.1 Methods of base class CDataType

The class is inherited of class CDataType [3.14] and supports all methods of this base class.

3.17.2 Methods of essential type objects

The class supports the set of Methods of essential type objects [2.7] for the type `unsigend char`.

3.18 class CDataTypeShort

The class represents the type `short`, was used within parsed DCD.

3.18.1 Methods of base class CDataType

The class is inherited of class CDataType [3.14] and supports all methods of this base class.

3.18.2 Methods of essential type objects

The class supports the set of Methods of essential type objects [2.7] for the type `short`.

3.19 class CDataTypeUnsignedShort

The class represents the type `unsigned short`, was used within parsed DCD.

3.19.1 Methods of base class CDataType

The class is inherited of class CDataType [3.14] and supports all methods of this base class.

3.19.2 Methods of essential type objects

The class supports the set of Methods of essential type objects [2.7] for the type `unsigend short`.

3.20 class CDataTypeLong

The class represents the type `long`, was used within parsed DCD.

3.20.1 Methods of base class CDataType

The class is inherited of class CDataType [3.14] and supports all methods of this base class.

3.20.2 Methods of essential type objects

The class supports the set of Methods of essential type objects [2.7] for the type `long`.

3.21 class CDataTypeUnsignedLong

The class represents the type `unsigned long`, was used within parsed DCD.

3.21.1 Methods of base class CDataType

The class is inherited of class CDataType [3.14] and supports all methods of this base class.

3.21.2 Methods of essential type objects

The class supports the set of Methods of essential type objects [2.7] for the type `unsigend long`.

3.22 class CDataTypeFloat

The class represents the type `float`, was used within parsed DCD.

3.22.1 Methods of base class CDataType

The class is inherited of class CDataType [3.14] and supports all methods of this base class.

3.22.2 Methods of essential type objects

The class supports the set of Methods of essential type objects [2.7] for the type `float`.

3.23 class CDataTypeDouble

The class represents the type `double`, was used within parsed DCD.

3.23.1 Methods of base class CDataType

The class is inherited of class CDataType [3.14] and supports all methods of this base class.

3.23.2 Methods of essential type objects

The class supports the set of Methods of essential type objects [2.7] for the type `double`.

3.24 class CDataTypeEnum

The class represents an enumeration, was used within parsed DCD.

3.24.1 Methods of base class CDataType

The class is inherited of class CDataType [3.14] and supports all methods of this base class.

3.24.2 GetEntryNameByValue

Declaration:

```
const char* GetEntryNameByValue(short nValue)
```

Parameter `nValue`:

Input parameter for value to search for.

Return:

Name of the enumeration member with specified value.
`NULL` if value is not available in the enumeration.

3.24.3 GetLocalName

Declaration:

```
const char*  GetLocalName()
```

Return:

Name of the enumeration.

3.24.4 `GetFirstMember`

Declaration:

```
bool GetFirstMember(const CEnumEntry *&rpoEnumEntry)
```

Parameter `rpoEnumEntry`:

Output parameter for the first entry in the enumeration.

Return:

`true`	Output parameter is set
`false`	Output parameter is not set

3.24.5 GetNextMember

There are two possibilities to use this method.

- Use to previous Enum Entry (parameter 2) as iterator to get the next.

  This way is save for multithreaded actions.

- Use the internal iterator to get the next Enum Entry (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextMember(
    const CEnumEntry *&rpoNextEnumEntry,
    const CEnumEntry *poPreviousEnumEntry = NULL )
```

Parameter `rpoNextEnumEntry`:

Output parameter for the next entry in the enumeration.

Parameter `poPreviousEnumEntry`:

Input parameter of the previous entry in the enumeration.

Return:

`true`	Output parameter is set.
`false`	Output parameter is not set.

3.25 class CDataTypeStruct

The class represents a structure, was used within parsed DCD.

3.25.1 Methods of base class CDataType

The class is inherited of class CDataType [3.14] and supports all methods of this base class.

3.25.2 GetLocalName

Declaration:

```
const char*  GetLocalName()
```

Return:

Name of the structure.

3.25.3 GetFirstElement

Declaration:

```
bool GetFirstElement (const CElement *&rpoElement  )
```

Parameter `rpoElement`:

Output parameter for the first element.

Return:

`true`	Output parameter is set
`false`	Output parameter is not set

3.25.4 `GetNextElement`

There are two possibilities to use this method.

- Use to previous Structure (parameter 2) as iterator to get the next.

  This way is save for multithreaded actions.

- Use the internal iterator to get the next Structure (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextElement(
    const CElement *&rpoNextElement,
    const CElement *poPreviousElement = NULL )
```

Parameter `rpoNextElement`:

Output parameter for the next element.

Parameter `poPreviousElement`:

Input parameter of the previous element.

Return:

`true`	Output parameter is set
`false`	Output parameter is not set

3.25.5 `GetElementByName`

Declaration:

```
bool GetElementByName (
    const CElement *&rpoElement,
    const char* szElementName )
```

Parameter `rpoElement`:

Output parameter for the element.

Parameter `szElementName`:

Input parameter for the element name to search for.

Return:

`true`	Output parameter is set.
`false`	Element name is not a member of this Structure.

3.26 `class CDataTypeUnion`

The class represents an union, was used within parsed DCD.

3.26.1 `Methods of base class CDataType`

The class is inherited of class CDataType [3.14] and supports all methods of this base class.

3.26.2 GetLocalName

Declaration:

```
const char*  GetLocalName()
```

Return:

Name of the union.

3.26.3 GetSwitchType

Declaration:

```
bool GetSwitchType (const CDataType *&rpoDataType) const;
```

Parameter `rpoDataType`:

Output parameter for the switch data type

Return:

`true`	Output parameter is set
`false`	Output parameter is not set

3.26.4 `GetFirstMember`

Declaration:

```
bool GetFirstMember(const CUnionElement *&rpoUnionElement )
```

Parameter `rpoUnionElement`:

Output parameter for the first union element.

Return:

`true`	Output parameter is set
`false`	Output parameter is not set

3.26.5 `GetNextMember`

There are two possibilities to use this method.

- Use to previous Union (parameter 2) as iterator to get the next.

  This way is save for multithreaded actions.

- Use the internal iterator to get the next Union (without parameter 2).

  This way is unsafe for multithreaded actions!

Declaration:

```
bool GetNextMember(
    const CUnionElement *&rpoNextElement,
    const CUnionElement *poPreviousUnionElement = NULL )
```

Parameter `rpoNextElement`:

Output parameter for the next element.

Parameter `poPreviousUnionElement`:

Input parameter of the previous element.

Return:

`true`	Output parameter is set
`false`	Output parameter is not set

3.26.6 `GetDefaultMember`

Declaration:

```
bool GetDefaultMember (const CUnionElement *&rpoUnionElement)
```

Parameter `rpoUnionElement`:

Output parameter for the default element

Return:

`true`	Output parameter is set
`false`	Output parameter is not set

3.26.7 `GetMemberByName`

Declaration:

```
bool GetElementByName(
    const CUnionElement *&rpoUnionElement,
    const char* szUnionElementName )
```

Parameter `rpoUnionElement`:

Output parameter for the union element.

Parameter `szUnionElementName`:

Input parameter for the union element name to search for.

Return:

`true`	Output parameter is set.
`false`	Element name is not a member of this union.

3.27 class CDataTypeSequence

The class uses the internal structure TSequenceManager for dynamic memory management. The structure is the same like it is stored in the Device Driver to represent a memory segment.

```
typedef struct
{
    unsigned long m_ulMaxSize;
    unsigned long m_ulCurrentSize;
    void*         m_pData;
} TSequenceManager;
```

The method GetMaxSizeOffset, GetCurrentSizeOffset and GetDataPointerOffset return the offset of the members of this structure.

3.27.1 Methods of base class CDataType

The class is inherited of class CDataType [3.14] and supports all methods of this base class.

3.27.2 GetMaxSizeOffset

Declaration:

```
unsigned long GetMaxSizeOffset()
```

3.27.3 GetCurrentSizeOffset

Declaration:

```
unsigned long GetCurrentSizeOffset()
```

3.27.4 GetDataPointerOffset

Declaration:

```
unsigned long GetDataPointerOffset()
```

3.27.5 GetBaseType

Declaration:

```
bool GetBaseType (const CDataType *&rpoDataType)
```

Parameter `rpoDataType`:

Output parameter for Base Data Type.

Return:

`true`	Output parameter is set.
`false`	Output parameter is not set.

3.27.6 GetMaxLength

This method returns the maximum length of the sequence.

Declaration:

```
unsigned long GetMaxLength()
```

Return:

Maximum length of Sequence.

`0` if the Sequence is an Unlimited Sequence.

3.27.7 GetSize

This method returns the current size in bytes of the sequence.

Declaration:

```
unsigned long GetSize()
```

3.27.8 GetMaxSize

This method returns the maximum size in bytes of the sequence.

Declaration:

```
unsigned long GetMaxSize()
```

3.27.9 IsElementarType

Declaration:

```
bool IsElementarType() const
```

3.27.10 IsStringType

Declaration:

```
bool IsStringType() const
```

3.28 class CDataTypeArray

The class represents an array, was used within parsed DCD.

3.28.1 Methods of base class CDataType

The class is inherited of class CDataType [3.14] and supports all methods of this base class.

3.28.2 GetBaseType

Declaration:

```
bool GetBaseType (const CDataType *&rpoDataType)
```

Parameter rpoDataType:

Output parameter for Base Data Type.

Return:

`true`	Output parameter is set.
`false`	Output parameter is not set.

3.28.3 GetSize

This method returns the current size in bytes of the Array.

Declaration:

```
unsigned long GetSize()
```

3.28.4 GetMaxNumberOfElements

Declaration:

```
unsigned long  GetMaxNumberOfElements()
```

Return:

Number of elements in array.

3.28.5 IsElementarType

Declaration:

```
bool IsElementarType() const
```

3.28.6 IsStringType

Declaration:

```
bool IsStringType() const
```

3.29 class CDataTypeInterfaceRef

The class represents the data type of Interfaces that are used as Operation parameter.

```
typedef struct
{
    unsigned long m_ulID;
    unsigned long m_ulHandle;
} CInterfaceRef;
```

3.29.1 Methods of base class CDataType

The class is inherited of class CDataType [3.14] and supports all methods of this base class.

3.29.2 GetInterfaceIDOffset

Declaration:

```
unsigned long GetInterfaceIDOffset ()
```

Return:

Offset of the interface id.

3.29.3 GetInterfaceHandleOffset

Declaration:

```
unsigned long         GetInterfaceHandleOffset ()
```

Return:

Offset of the interface handle.

3.29.4 GetAttParamIDOffset

Declaration:

```
unsigned long GetAttParamIDOffset() const;
```

3.29.5 GetAttParamHandleOffset

Declaration:

```
unsigned long GetAttParamHandleOffset()  const;
```

3.29.6 `IsInterfaceReference`

Declaration:

```
bool IsInterfaceReference() const;
```

3.29.7 `IsAttParamReference`

Declaration:

```
bool IsAttParamReference() const;
```

3.29.8 `GetSize`

Declaration:

```
virtual unsigned long GetSize() const;
```

3.30 `class CEnumEntry`

The class represents an enumerator of an enumeration of `class CDataTypeEnum` [3.24], was used within parsed DCD.

3.30.1 `Methods for declaration information`

The class supports the set of Methods for declaration information [2.8].

3.30.2 GetName

Declaration:

```
const char* GetName()
```

Return:

Name of the enumeration member.

3.30.3 GetValue

Declaration:

```
short GetValue()
```

Return:

Value of the enumeration member.

3.31 class CElement

The class represents an element of a structure of `class CDataTypeStruct` [3.25], was used within parsed DCD.

3.31.1 Methods to access content of DITs

The class supports the set of Methods to access content of DITs [2.6].

3.31.2 Methods for declaration information

The class supports the set of Methods for declaration information [2.8].

3.31.3 GetName

Declaration:

```
const char* GetName()
```

Return:

Name of the structure member.

3.31.4 GetDataType

Declaration:

```
Bool GetDataType (const CDataType *&rpoDataType)
```

Parameter `rpoDataType`:

Output parameter for data type of the structure member.

Return:

`true`	Output parameter is set.
`false`	Output parameter is not set.

3.31.5 GetOffset

Declaration:

```
unsigned long GetOffset()
```

Return:

Offset of the structure member.

3.32 class CUnionElement

The class represents an element of a structure of `class CDataTypeUnion` [3.26], was used within parsed DCD.

3.32.1 Methods to access content of DITs

The class supports the set of Methods to access content of DITs [2.6].

3.32.2 Methods for declaration information

The class supports the set of Methods for declaration information [2.8].

3.32.3 GetName

Declaration:

```
const char* GetName()
```

Return:

Name of the union member.

3.32.4 GetDataType

Declaration:

```
Bool GetDataType (const CDataType *&rpoDataType)
```

Parameter `rpoDataType`:

Output parameter for data type of the union member.

Return:

`true`	Output parameter is set
`false`	Output parameter is not set

3.32.5 GetSwitchValue

Declaration:

```
Long GetSwitchValue()
```

Return:

Switch value.

3.32.6 IsDefault

Declaration:

```
Bool IsDefault()
```

Return:

`true`	Union element is default member.
`false`	Is not the default member.