ASAM GDI DITParser

API Documentation

rd electronic GmbH

location Zwickau

Newtonstraße 12a

08060 Zwickau

Tel. 0375-447990-70

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

Document Number:	DITParser-APIDoc
Issue:	1.1.0.2
Status:	releasedt
Last Modification:	2021-03-30
Created on:	2017-12-10
Authors:	KSc (rde) SBa (rde)

Important Notice

Document Change History

Version	Date	Changed by	Modification
1.1.0.0	2017-12-10	SBa	Created Original “GDI DITParser43 UserMan”
1.1.0.1	2017-12-22	SBa	Correction of technical terms
1.1.0.2	2021-03-30	SBa	Update company information

Document Approval Sheet

Document Number:	DITParser-APIDoc
Title:	ASAM GDI DITParser
Subtitle:	API Documentation
Issue:	1.1.0.2
Date:	2021-03-30
Status:	releasedt

Contents

1 General

1.1 Identification

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

The described interface is based on DITParser v1.2.5.2.

The DITParser API allows the user of the DITParser to get information about the contained DIT definitions of the parsed DIT.

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

1.2 Introduction

The DITParser API allows the user of the DITParser to get information about the text messages in DIT files. The DITParser is able to work without a DCDParser.

The DITParser API is contained within the namespace “DITParser”.

2 DITParser API

To start the DCDParser it is necessary to call the method Parse [2.1.1] of the class CDITModule [2.1].

The successful parse process is required for using of all other classes and theirs methods.

2.1 class CDITModule

2.1.1 `Parse`

The method parses the specified DIT file and builds an internal structure with image of the parsed DIT file. If the parameter for DIT language is used, then the DITParser search for the adapted standard DIT file and parse this file in the same internal structure.

This method have to be called before using over methods of the DITParser API.

Declaration:

```
CParseStatus* Parse(
    const char* szDITFile,
    const char* pszComErrorDIT = NULL,
    const char* pszLanguage = NULL )
```

Parameter `szDITFile`:

Name of the DIT file to parse.

Parameter `pszComErrorDIT`:

Path to the Com Error DIT file

Parameter `pszLanguage`:

Do not use this. For future implementation.

Return:

Pointer to object with information about the parse process.

2.1.2 `GetParserVersion`

Declaration:

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

Return:

Version of the used DITParser.

2.1.3 `GetTextByIDs`

If you want to get information about a DIT file using the DIT access IDs of a parsed DCD file, then it is recommendable to use the method GetTextDITAccess of the DITParser. The DITParser uses the DITParser with a user-friendly interface, so what you can use the parsed DITAccess structure as parameter for method GetTextByDITAccess.

Declaration:

```
const char* GetTextByIDs(
    unsigned short   unAreaID,
    unsigned short   unSubAreaID,
    unsigned short   unTextID,
    bool*            pbIDValid = NULL )
```

Parameter `unAreaID`:

Area ID of the text.

Parameter `unSubAreaID`:

SubArea ID of the text.

Parameter `unTextID`:

Text ID of the text.

Parameter `pbIDValid`:

Output parameter which signals the validity of the IDs.

`true`	Text for IDs is valid
`false`	Text for the IDs is invalid

Return:

Text which is specified by the IDs.

2.1.4 `GetTextByResult`

If you are using the DITParser, then you can use the method GetTextByResult with a more user friendly interface. The DITParser allows to use the result structure as parameter for the method GetTextByResult.

Declaration:

```
const char* GetTextByResult(
    short            nQual,
    short            nGrade,
    short            nCode,
    short            nReturnValue,
    char*            pszzInsText,
    char*            pcAllocMemory,
    unsigned long    ulMemorySize,
    bool*            pbValid  = NULL )
```

Parameter `nQual`:

Qual.

Parameter `nGrade`:

Grade.

Parameter `nCode`:

Code.

Parameter `nReturnValue`:

Return Code.

Parameter `pszzInsText`:

Double terminated string with replace strings.

Parameter `pcAllocMemory`:

Poiner to allocated memory for the text.

Parameter `ulMemorySize`:

Size of the allocated memory.

Parameter `pbValid`:

Output parameter which signals the validityof the IDs.

`true`	Text for result is valid.
`false`	Text for the result is invalid.

Return:

Text which is specified by the result.

2.1.5 `GetNumberOfAreas`

Declaration:

```
unsigned long GetNumberOfAreas()
```

Return:

Number of available areas.

2.1.6 `GetUserAreaByID`

Declaration:

```
bool GetUserAreaByID(const CArea *&rpoArea, unsigned short
unAreaID)
```

Parameter `rpoArea`:

Output parameter for the UserArea.

Parameter `unAreaID`:

Input parameter for the UserArea ID to search for.

Return:

`true`	UserArea with specified ID is available.
`false`	UserArea ID is not valid.

2.1.7 GetUserAreaByName

Declaration:

```
bool GetUserAreaByName(
    const CArea *&rpoArea, const char* szAreaName)
```

Parameter `rpoArea`:

Output parameter for the UserArea.

Parameter `szAreaName`:

Input parameter for the UserArea name to search for.

Return:

`true`	UserArea with specified name is available
`false`	UserArea name is not valid

2.1.8 GetFirstUserArea

Declaration:

```
bool GetFirstUserArea(const CArea *&rpoArea)
```

Parameter `rpoArea`:

Output parameter for the UserArea.

Return:

`true`	Output parameter is initialized
`false`	There is no UserArea available in the DIT

2.1.9 GetNextUserArea

Declaration:

```
bool GetNextUserArea(
    const CArea *&rpoNextArea, const CArea *poPreviousArea)
```

Parameter `rpoNextArea`:

Output parameter for the next UserArea.

Parameter `poPreviousArea`:

Input parameter with the last UserArea.

Return:

`true`	Output parameter for the next is initialized.
`false`	There is no next UserArea available in the DIT.

2.1.10 GetStdAreaByID

Declaration:

```
bool GetStdAreaByID(const CArea *&rpoArea, unsigned short
unAreaID)
```

Parameter `rpoArea`:

Output parameter for the StdArea.

Parameter `unAreaID`:

Input parameter for the StdArea ID to search for.

Return:

`true`	StdArea with specified ID is available.
`false`	StdArea ID is not valid.

2.1.11 `GetStdAreaByName`

Declaration:

```
bool GetStdAreaByName(
    const CArea *&rpoArea, const char* szAreaName)
```

Parameter `rpoArea`:

Output parameter for the StdArea.

Parameter `szAreaName`:

Input parameter for the StdArea name to search for.

Return:

`true`	StdArea with specified name is available
`false`	StdArea name is not valid

2.1.12 `GetFirstStdArea`

Declaration:

```
bool GetFirstStdArea(const CArea *&rpoArea)
```

Parameter `rpoArea`:

Output parameter for the StdArea.

Return:

`true`	Output parameter is initialized
`false`	There is no StdArea available in the DIT

2.1.13 GetNextStdArea

Declaration:

```
bool GetNextStdArea(
    const CArea *&rpoNextArea, const CArea *poPreviousArea)
```

Parameter `rpoNextArea`:

Output parameter for the next StdArea.

Parameter `poPreviousArea`:

Input parameter with the last StdArea.

Return:

`true`	Output parameter for the next is initialized.
`false`	There is no next StdArea available in the DIT.

2.1.14 GetDeclarationFile

Returns the declaration DIT file of this DIT Module.

Declaration:

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

Return:

File path.

2.1.15 GetNumberOfParsedFiles()

Returns the number of parsed DIT files that are parsed for this DIT Module.

Declaration:

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

Return:

Number of files.

2.1.16 GetParsedFileByIndex

Returns parsed file with given index.

Declaration:

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

Return:

File path.

2.2 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.

The DITParser 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.

2.2.1 GetReturnCode

Get the return code of the parse process Parse [2.1.1].

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

2.2.2 GetMessage

Get the message of the parse process Parse [2.1.1].

Declaration:

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

Return:

Message text.

2.2.3 GetErrorMessage

Get the error message of the parse process Parse [2.1.1].

Declaration:

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

Return:

Error message text.

2.2.4 GetErrorFile

Get file name that contains the parse error.

Declaration:

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

Return:

File name.

2.2.5 GetErrorLine

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

Declaration:

```
unsigned long GetErrorLine()
```

Return:

Line number of error.

2.2.6 GetNumberOfWarnings

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

Declaration:

```
unsigned long GetNumberOfWarnings()
```

Return:

Number of warnings.

2.2.7 GetWarningMessageByNumber

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

Declaration:

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

Return:

Message text.

2.2.8 GetWarningFileByNumber

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

Declaration:

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

Return:

File name.

2.2.9 GetWarningLineByNumber

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

Declaration:

```
unsigned long GetWarningLineByNumber()
```

Return:

Line within file.

2.3 class CArea

2.3.1 GetName

Declaration:

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

Return:

Name of the area.

2.3.2 GetID

Declaration:

```
unsigned long GetID()
```

Return:

ID of the area.

2.3.3 GetNumberOfSubAreas

Declaration:

```
unsigned long GetNumberOfSubAreas()
```

Return:

Number of sub area in the current area.

2.3.4 GetTextByIDs

This method of CArea is able to return text messages of the DIT file related to a specified SubArea- and Text ID.

Declaration:

```
const char* GetTextByIDs (
    unsigned short unSubAreaID,
    unsigned short unTextID,
    bool* pbIDValid = NULL )
```

Return:

Text which is specified by the sub area ID and text ID.

2.3.5 GetSubAreaByID

Declaration:

```
bool GetSubAreaByID(
    const CSubArea *&rpoSubArea,
    unsigned short unSubAreaID)
```

Parameter `rpoSubArea`:

Output parameter for the SubArea.

Parameter `unSubAreaID`:

Input parameter for the SubArea ID to search for.

Return:

`true`	SubArea with specified ID is available.
`false`	SubArea ID is not valid.

2.3.6 GetSubAreaByName

Declaration:

```
bool GetSubAreaByName(
    const CSubArea *&rpoSubArea, const char* szAreaName)
```

Parameter `rpoSubArea`:

Output parameter for the SubArea.

Parameter `szAreaName`:

Input parameter for the SubArea name to search for.

Return:

`true`	SubArea with specified name is available
`false`	SubArea name is not valid

2.3.7 GetFirstSubArea

Declaration:

```
bool GetFirstSubArea(const CSubArea *&rpoSubArea)
```

Parameter `rpoSubArea`:

Output parameter for the SubArea.

Return:

`true`	Output parameter is initialized
`false`	There is no SubArea available in the DIT

2.3.8 GetNextSubArea

Declaration:

```
bool GetNextSubArea(
    const CSubArea *&rpoNextSubArea,
    const CSubArea *poPreviousSubArea)
```

Parameter `rpoNextSubArea`:

Output parameter for the next SubArea.

Parameter `poPreviousSubArea`:

Input parameter with the last SubArea.

Return:

`true`	Output parameter for the next is initialized.
`false`	There is no next SubArea available in the DIT.

2.4 class CSubArea

2.4.1 GetName

Declaration:

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

Return:

Name of the sub area.

2.4.2 GetID

Declaration:

```
unsigned long GetID()
```

Return:

ID of the sub area.

2.4.3 GetNumberOfTexts

Declaration:

```
unsigned long GetNumberOfTexts()
```

Return:

Number of texts in the current sub area.

2.4.4 GetTextByID

Declaration:

```
const char* GetTextByIDs (
    unsigned short unTextID,
    bool* pbIDValid = NULL )
```

Return:

Text which is specified by text ID.

2.4.5 GetFirstText

Declaration:

```
const char* GetFirstText(
    unsigned short &runTextID, bool* pbIDValid = NULL )
```

Parameter `runTextID`:

Output parameter for the Text ID.

Parameter `pbIDValid`:

Output parameter which signals the validity of the IDs.

`true`	Text for result is valid
`false`	Text for the result is invalid

Return:

First text specified in the sub area.

2.4.6 `GetNextText`

Declaration:

```
const char* GetNextText(
    unsigned short &runNextTextID,
    unsigned short unPreviousTextID,
    bool* pbIDValid = NULL )
```

Parameter `runNextTextID`:

Output parameter for the next Text ID.

Parameter `unPreviousTextID`:

Input parameter for the last Text ID.

Parameter `pbIDValid`:

Output parameter which signals the validity of the IDs.

`true`	Text for result is valid.
`false`	Text for the result is invalid.

Return:

Next text specified in the sub area.