

# **IDoc Class Library (BC-FES-AIT)**



HELP·BCFESDED

**Release 4.6C**



## Copyright

© Copyright 2001 SAP AG. Alle Rechte vorbehalten.

Weitergabe und Vervielfältigung dieser Publikation oder von Teilen daraus sind, zu welchem Zweck und in welcher Form auch immer, ohne die ausdrückliche schriftliche Genehmigung durch SAP AG nicht gestattet. In dieser Publikation enthaltene Informationen können ohne vorherige Ankündigung geändert werden.

Die von SAP AG oder deren Vertriebsfirmen angebotenen Software-Produkte können Software-Komponenten auch anderer Software-Hersteller enthalten.

Microsoft<sup>®</sup>, WINDOWS<sup>®</sup>, NT<sup>®</sup>, EXCEL<sup>®</sup>, Word<sup>®</sup>, PowerPoint<sup>®</sup> und SQL Server<sup>®</sup> sind eingetragene Marken der Microsoft Corporation.

IBM<sup>®</sup>, DB2<sup>®</sup>, OS/2<sup>®</sup>, DB2/6000<sup>®</sup>, Parallel Sysplex<sup>®</sup>, MVS/ESA<sup>®</sup>, RS/6000<sup>®</sup>, AIX<sup>®</sup>, S/390<sup>®</sup>, AS/400<sup>®</sup>, OS/390<sup>®</sup> und OS/400<sup>®</sup> sind eingetragene Marken der IBM Corporation.

ORACLE<sup>®</sup> ist eine eingetragene Marke der ORACLE Corporation.

INFORMIX<sup>®</sup>-OnLine for SAP und Informix<sup>®</sup> Dynamic Server<sup>™</sup> sind eingetragene Marken der Informix Software Incorporated.

UNIX<sup>®</sup>, X/Open<sup>®</sup>, OSF/1<sup>®</sup> und Motif<sup>®</sup> sind eingetragene Marken der Open Group.

HTML, DHTML, XML, XHTML sind Marken oder eingetragene Marken des W3C<sup>®</sup>, World Wide Web Consortium, Massachusetts Institute of Technology.

JAVA<sup>®</sup> ist eine eingetragene Marke der Sun Microsystems, Inc.

JAVASCRIPT<sup>®</sup> ist eine eingetragene Marke der Sun Microsystems, Inc., verwendet unter der Lizenz der von Netscape entwickelten und implementierten Technologie.

SAP, SAP Logo, R/2, RIVA, R/3, ABAP, SAP ArchiveLink, SAP Business Workflow, WebFlow, SAP EarlyWatch, BAPI, SAPPHIRE, Management Cockpit, mySAP.com Logo und mySAP.com sind Marken oder eingetragene Marken der SAP AG in Deutschland und vielen anderen Ländern weltweit. Alle anderen Produkte sind Marken oder eingetragene Marken der jeweiligen Firmen.

## Symbole

Symbol	Bedeutung
	Achtung
	Beispiel
	Hinweis
	Empfehlung
	Syntax

## Inhalt

<b>IDoc Class Library (BC-FES-AIT)</b> .....	<b>7</b>
<b>IDoc Class Library</b> .....	<b>8</b>
<b>Introduction</b> .....	<b>9</b>
Overview.....	10
Who Should Use this Help File.....	11
System Requirements.....	12
Building and Running Applications.....	13
R/3 Release-Dependent Issues.....	14
General Class Design Principles.....	15
General Descriptions of the Classes.....	16
CIDoc Class Summary.....	17
CIDocFieldIterator Class Summary.....	18
CIDocIterator Class Summary.....	19
Three Ways to Use CIDocIterator.....	20
Using GetChild.....	21
Using NextRepetition or operator++ with count.....	22
Using NextRepetition or operator++ without count.....	23
CIDocLink Class Summary.....	24
CIDocMetaDataTree Class Summary.....	25
CIDocOutboundApp Class Summary.....	26
CIDocOutboundFunc Class Summary.....	27
CIDocTID Class Summary.....	28
Basic Logic for Using the Class Library.....	29
Sending IDocs Inbound to R/3.....	30
Receiving IDocs Outbound from R/3.....	31
<b>Class Library Reference</b> .....	<b>32</b>
The CIDoc Class.....	33
AppendSegment.....	34
GetControlField.....	35
GetIDocNumber.....	36
GetIDocType.....	37
GetMessageType.....	38
GetPartner.....	39
GetPort.....	40
GetRelease.....	41
SetControlField.....	42
SetIDocNumber.....	43
WriteIDocToFile.....	44
The CIDocFieldIterator Class.....	45
CIDocFieldIterator (copy constructor).....	47
First.....	48
GetFieldInfo.....	49
GetFloatValue.....	50

GetInternalType .....	51
GetIntValue .....	52
GetLength .....	53
GetName .....	54
GetOffset .....	55
GetStringValue .....	56
IsEnd .....	57
IsFirst .....	58
Last .....	59
operator ++ (pre-increment version) .....	60
operator ++ (post-increment version) .....	61
operator -- (pre-decrement version) .....	62
operator -- (post-decrement version) .....	63
operator = (floating-point) .....	64
operator = (string) .....	65
operator = (const char*) .....	66
operator = (integer) .....	67
operator = (assignment from another iterator) .....	68
operator [] (by index) .....	69
operator [] (by name) .....	70
The CIDocIterator Class .....	71
CIDocIterator .....	73
Field (write to field) .....	74
Field (read from field) .....	75
GoTop .....	76
NextSibling .....	77
NextRepetition .....	78
FirstChild .....	79
GetChild .....	80
GetName .....	81
GetChildCount .....	82
GetSegmentCount .....	83
GetFirstChildWithName .....	84
operator ++ .....	85
IsThereMoreOfThis .....	86
The CIDocLink Class .....	87
AddIDoc (stored in file) .....	88
AddIDoc (created from scratch) .....	89
AddIDoc (from scratch, for extension types) .....	90
CIDocLink .....	91
CIDocLink (with user-defined outbound application) .....	92
CopyMetaDataTree .....	93
ExitWaitLoop .....	94
GetConnection .....	95
GetIDoc .....	96

GetIDocCount .....	97
GetIDocsFromR3 .....	98
GetTIDMgr .....	99
RemoveAllIDocs .....	100
ResendFailedIDocs.....	101
SendIDocsToR3.....	102
The CIDocMetaDataTree Class.....	103
CIDocMetaDataTree (copy constructor).....	104
FindChildSegment .....	105
FindField .....	106
FindSegment.....	107
GetSegmentCount .....	108
operator [].....	109
The CIDocOutboundApp Class.....	110
CIDocOutboundApp.....	111
OnCheckID .....	112
OnCommit.....	113
OnConfirmID .....	114
OnIdle.....	115
OnRollBack .....	116
The CIDocOutboundFunc Class .....	117
CIDocOutboundFunc .....	118
Process .....	119
The CIDocDataSegment Class.....	120
The CIDocFieldInfo Class .....	122
The CIDocTID Class .....	123
CIDocTID .....	124
CheckTID .....	125
UpdateTID.....	126
FindFailedTID .....	127
WriteTID .....	128
LockTID.....	129
UnlockTID .....	130
The CIDocTrace Class.....	131
SetTrace.....	132
Trace .....	133
TraceBlankLines .....	134
TraceIDOCError.....	135
TraceRFCError .....	136
TraceTime.....	137

## IDoc Class Library (BC-FES-AIT)

---

**IDoc Class Library**

## **IDoc Class Library**

The following topics are available to assist programmers using the IDoc Class Library. This document explains how the classes relate to one another and how to work with the class library.

[Introduction \[Seite 9\]](#)

[Class Library Reference \[Seite 32\]](#)

For more information on using IDocs, see also [IDoc Interface / Electronic Data Interchange \[Extern\]](#).

## Introduction

[Overview \[Seite 10\]](#)

[Who Should Use this Help File \[Seite 11\]](#)

[R/3 Release-Dependent Issues \[Seite 14\]](#)

[Platform-Dependent Issues \[Seite 12\]](#)

[General Class Design Principles \[Seite 15\]](#)

---

**Overview**

## Overview

The IDoc class library offers C++ classes that encapsulate IDoc processing functionality. IDocs are a document transfer technique that allows external applications to send documents back and forth when communicating with R/3 via Remote Function Calls (RFC).

The IDoc Class Library allows you to write C++ code that:

- Makes transactional RFC calls to R/3 to send IDocs inbound to R/3.
- Acts as a transactional RFC server to receive outbound IDocs from R/3.
- Creates IDocs segment by segment.
- Navigates through the IDoc tree structure to access the IDoc segments. Segments are accessed by name.
- Reads data from and writes data to fields in segments. Fields are accessed by name; data type conversions are done automatically.

## Who Should Use this Help File

This document assumes that the reader has a basic knowledge of:

- integration of desktop applications with R/3 using Remote Function Call (RFC), especially the transactional RFC (tRFC).
- the fundamentals of IDoc structures such as hierarchy levels, control segments, data segments, segment groups, parent segments, repetition of segments and repetition of segment groups, and segment fields.

---

**System Requirements**

## System Requirements

### Development Requirements

To create applications using the IDoc Class Library you need the following:

- Windows NT (Service Pack 3) or Windows 95 operating system
- Visual Studio 97 (Visual C++ 5.0) with Visual Studio Service Pack 3

### Run-time Requirements

The end user of an applications using the IDoc Class Library needs the same platform:

- Windows NT (Service Pack 3) or Windows 95 operating system
- Visual Studio 97 (Visual C++ 5.0) with Visual Studio Service Pack 3

## Building and Running Applications

### Required Compiler Options for *Debug* Version

```
/nologo /MDd /W3 /GX /Z7 /Od /I "<directory-where-IDoc-Class-Library's-headers-are-located>" /I  
"<directory-where-RFC-Class-Library's-headers-are-located>" /I "<directory-where-RFC-API's-headers-are-  
located>" /D "WIN32" /D "_DEBUG" /D "_CONSOLE" /D "_MBCS" /Fp"<file-path-of-pre-compiled-header>"  
/YX /Fo"Debug/" /Fd"Debug/" /FD /c
```

### Required Compiler Options for *Release* Version

```
/nologo /ML /W3 /GX /O2 /Od /I "<directory-where-IDoc-Class-Library's-headers-are-located>" /I "<directory-  
where-RFC-Class-Library's-headers-are-located>" /I "<directory-where-RFC-API's-headers-are-located>" /D  
"WIN32" /D "NDEBUG" /D "_CONSOLE" /D "_MBCS" /Fp"<file-path-of-pre-compiled-header>" /YX  
/Fo"Release/" /Fd"Release/" /FD /c
```

## R/3 Release-Dependent Issues

# R/3 Release-Dependent Issues

## Metadata

The IDoc Class Library uses metadata to parse the segments and fields for the various types of IDocs. When the application program runs, the IDoc Class Library determines where to obtain the necessary metadata for the types of IDocs being processed.

If your application programs transfers IDocs to and from R/3 Systems earlier than release 3.1G (up to 3.0D), you must have a metadata file in the same directory as the application program. You can obtain this file in the following manner:

1. Log on to the R/3 system with which your program will be transferring IDocs.
2. Use Transaction SE38, and execute the program RSEIDOC5.
3. In the two edit boxes labeled *Intermediate Document type*, enter only one IDoc type.
4. Click the 2 check boxes labeled *Display structures* and *Display segment fields*. Leave *Display documentation* unchecked.
5. Click the Execute button and wait for display of segment and field information.
6. Pull down the *List* menu and choose the *Download* menu item. When a dialog box appears, leave the *Unconverted* radio button checked. Press *Continue* button.
7. Enter the full path and file name in the *File name* edit box.

The path name must be the directory where the executable file for the application is located. The file name must have the format: **<TYPE>\_<REL>.mtd**, where **<TYPE>** is the IDoc type, **<REL>** is the release and **mtd** is the file extension for metadata.

Examples are: "MATMAS02\_30D.mtd" and "ORDERS01\_30D.mtd". The characters in the file name must be in lower case on platforms where file names are case-sensitive.

## Function Modules for Sending IDocs Inbound

R/3 releases 3.0 and 3.1 use the RFC function module INBOUND\_IDOC\_PROCESS to receive IDocs, and call a function of the same name to send IDocs outbound from R/3 to an external application program which must implement that function.

In R/3 release 4.0, a second function module is available called IDOC\_INBOUND\_ASYNCHRONOUS. Any application that uses the IDoc Class Library to transfer IDocs inbound to R/3 4.0 must specify the transfer method. For outbound processing, the IDoc Class Library implements two RFC server functions, named INBOUND\_IDOC\_PROCESS and IDOC\_INBOUND\_ASYNCHRONOUS. To the application program, it is transparent which function has been called.

For sending IDocs inbound to a 4.0 R/3, the application program specifies the inbound method using one of the arguments in the constructor of the class CIDocLink. Your application program should use the enumerated data type INBOUND\_METHOD (defined in *idocglob.h*), to tell the IDoc Class Library which method is to be used when sending IDocs inbound to R/3.

## General Class Design Principles

The IDoc Class Library was implemented with the following objectives in mind:

- Minimum effort on the part of the application program to create, send, receive and process IDocs.
- Platform independence, and ANSI C++ compliance.
- Optimal execution speed.
- Minimum code size overhead.
- Protection for all data members of classes, by allowing access only through methods.

For additional information, see the following:

[General Descriptions of the Classes \[Seite 16\]](#)

[Basic Logic for Using the Class Library \[Seite 29\]](#)

---

**General Descriptions of the Classes**

## General Descriptions of the Classes

The classes offered in the class library are:

[CIDocLink Class Summary \[Seite 24\]](#)

[CIDoc Class Summary \[Seite 17\]](#)

[CIDocIterator Class Summary \[Seite 19\]](#)

[CIDocFieldIterator Class Summary \[Seite 18\]](#)

[CIDocMetaDataTree Class Summary \[Seite 25\]](#)

[CIDocDataSegment Class Summary \[Extern\]](#)

[CIDocFieldInfo Class Summary \[Extern\]](#)

[CIDocTID Class Summary \[Seite 28\]](#)

## CIDoc Class Summary

This class encapsulates one IDoc and offers functions to:

- Construct an IDoc from the content of a file.
- Read from and write to fields in the control segment of the IDoc.
- Append a segment of a specific type to the encapsulated IDoc, with checks to enforce compliance to the structure defined for the IDoc type.
- Write the encapsulated IDoc to a file.

---

**CIDocFieldIterator Class Summary**

## CIDocFieldIterator Class Summary

This class offers functions for the application program to iterate sequentially through all fields or randomly access any field of a an IDoc segment and obtain information such as name, offset, type and value of the fields.

An object of this class is created by an object of the CIDocIterator class. The newly created object of this class iterates through the fields of the IDoc segment pointed to by the parent CIDocIterator object. When the parent CIDocIterator object moves on to a different IDoc segment, the already-created CIDocFieldIterator object(s) do not move on to that IDoc segment along with the parent CIDocIterator object. These CIDocFieldIterator objects remain on the same IDoc segment through whose fields they were originally created to iterate.

The functions of this class implement the following functionalities:

- Pre- and post-increment operators for advancing to next field.
- Pre- and post-decrement operators for advancing to previous field.
- Randomly positioning to first field.
- Randomly positioning to last field.
- Returning the field information structure.
- Randomly positioning to the field specified by an index.
- Randomly positioning to the field specified by the field name.
- Returns name, offset, length and value (of types integer, string, floating-point).
- Returns internal type of the field.
- Returns flag indicating whether the iterator is positioned at the first or last field.
- Sets integer, string and floating-point value to field.

## CIDocIterator Class Summary

The objects of the CIDocIterator class are used for traversing an IDoc, and reading and modifying the fields in the segments of the IDoc. Within the context of the IDoc Class Library, a CIDocIterator object is always referred to as an iterator.

A CIDocIterator object must associate itself with a single IDoc at the time of its (the iterator's) construction. After that, you can use the iterator to traverse only that IDoc and no other.

The majority of the functions offered by the CIDocIterator class are for traversing the hierarchical tree structure of the IDoc. For example, you can position one CIDocIterator on a parent segment while using another CIDocIterator to advance through the child segments for that parent. If the child segments are themselves parent segments, you construct yet another CIDocIterator object to traverse the child segments at the next level down.

For information on traversing segment groups, see [Three Ways to Use CIDocIterator \[Seite 20\]](#).

The functions offered by the CIDocIterator class are:

- Reading the content of a field in the segment pointed at by the iterator.
- Modifying the content of a field in the segment pointed to by the iterator.
- Advancing the iterator to the next sibling segment within the same segment group.
- Advancing the iterator to the next identically-named sibling segment within the same segment group.
- Advancing the iterator to the first child segment when the iterator points to a parent segment.
- Retrieving the total number of child segments for a given parent segment.
- Retrieving a specific child segment when the iterator points to a parent segment.
- Retrieving the name of the segment pointed at by the iterator.
- Retrieving the total number of child segments with a specified name
- Querying whether there are any more identically-named sibling segments within the segment group.
- Creating CIDocFieldIterator objects for iterating through the fields in the current segment.

---

**Three Ways to Use CIDocIterator**

## Three Ways to Use CIDocIterator

The following three sections describe three basic ways to traverse a segment group. These examples use the E1MARAM and E1MAKTM segments of the MATMAS01 or MATMAS02 IDoc type, where E1MARAM is a parent segment and E1MAKTM is one of E1MARAM's children segments.

[Using GetChild \[Seite 21\]](#)

[Using NextRepetition or operator++ with count \[Seite 22\]](#)

[Using NextRepetition or operator++ without count \[Seite 23\]](#)

## Using GetChild

```
CIDoc& IDoc1 = Link.GetIDoc(index);
CIDocIterator it1(IDoc1), it2(IDoc1); // both iterator for IDoc1
it1.GoTop();
... // here is code that positions it1 on E1MARAM
int nChildren = it1.GetChildCount();
for(int i=0; i < nChildren; ++i)
{
    it2 = it1.GetChild(i);
    // if the segment is E1MAKTM, process it
    if(it2.GetName() == "E1MAKTM")
    {
        Lan = it2.Field("SPRAS"); // accesses
        SPRAS field
        ... // more code that accesses other fields
    }
}
```

## Using NextRepetition or operator++ with count

## Using NextRepetition or operator++ with count

```
CIDoc& IDoc1 = Link.GetIDoc(index);
CIDocIterator it1(IDoc1); // both iterator for IDoc1
it1.GoTop();
... // here is code that positions it1 on E1MARAM
int nMaktm = it1.GetSegmentCount("E1MAKTM");
it1 = it1.GetFirstChildWithName("E1MAKTM");
for(int i=0; i < nMaktm; ++i)
{
    Lan = it1.Field("SPRAS");
    ... // more code that accesses other fields
    it1.NextRepetition(); // or, ++it1;
}
```

## Using NextRepetition or operator++ without count

NOTE: the following sample code is best used with segments that are mandatory.

```
CIDoc& IDoc1 = Link.GetIDoc(index);
CIDocIterator it1(IDoc1),it2(IDoc1); // both iterator for IDoc1
it1.GoTop();
... // here is code that positions it1 on E1MARAM
// next call would throw an exception if there is no E1MAKTM,
// therefore this example is best used with mandatory segments
it2 = it1.GetFirstChildWithName("E1MAKTM");
do
{
    Ian = it2.Field("SPRAS");
    ... // more code that accesses other fields
    it2.NextRepetition(); // or, ++it2;
}
while(it2.IsThereMoreOfThis());
```

---

**CIDocLink Class Summary**

## CIDocLink Class Summary

This class is the all-encapsulating class responsible for:

- Waiting and receiving IDocs from R/3.
- Calling R/3 to send IDocs to it.
- Maintaining a list of IDocs.

A CIDocLink object is responsible for either an inbound or an outbound link. It also needs connection parameters (user and system parameters) for connecting to an R/3 system and a CIDocTID object to manage Transaction Ids (TIDs).

A CIDocLink object offers functions to:

- Send IDocs to R/3.
- Receive IDocs from R/3.
- Get the total number of received IDocs.
- Get a reference to a specific IDoc in a list of IDocs.
- Add an IDoc to the list of IDocs.
- Saves IDocs to a file when the inbound call fails.
- Re-sends IDocs saved from a failed call.
- Returns a copy of the metadata for a specified IDoc type.

## CIDocMetaDataTree Class Summary

This class encapsulates IDoc metadata. IDoc metadata is identified by IDoc type and its release version. For example, a CIDocMetaDataTree object may contain IDoc metadata for IDoc type MATMAS02, release 40B. IDoc metadata is stored as a tree, with a list of CIDocDataSegment objects containing metadata for an IDoc segment and each CIDocDataSegment object containing a list of CIDocFieldInfo objects containing metadata for the fields.

The application program may obtain its own copy of the metadata by calling `CIDocLink::CopyMetadataTree()`. This function returns pointer to a dynamically allocated CIDocMetaDataTree object. The application program must remember to delete this object after use since the ownership of the dynamically allocated object is transferred to the application program. The metadata objects owned by the IDoc Class Library remains private and are not accessible by the application program.

---

**CIDocOutboundApp Class Summary**

## CIDocOutboundApp Class Summary

The CIDocOutboundApp class is derived from CrfcTransApp. You can use CIDocOutboundApp as a base class for deriving user-defined classes and adding user-defined functionality. However, CIDocOutboundApp also offers basic implementation that can be used as is. This implementation lets you make quick prototypes of applications that accept IDocs from R/3. The basic implementation of this class includes the following:

- **OnCheckID:**  
Implements the pure virtual function CRfcTransApp::OnCheckID. OnCheckID writes the transaction ID (the TID created by the RFC library) to the default TID management file and sets the TID state to CREATED.
- **OnCommit:**  
Implements the pure virtual function CRfcTransApp::OnCommit by changing the state of the TID in question to EXECUTED.
- **OnConfirmID:**  
Implements the pure virtual function CRfcTransApp::OnConfirmID by changing the state of the TID in question to CONFIRMED.
- **OnRollBack:**  
Implements the pure virtual function CRfcTransApp::OnRollBack by changing the state of the TID in question to ROLLBACK.
- **OnIdle:**  
Implements the pure virtual function CRfcTransApp::OnIdle by sleeping for one second.

All of the functions above are declared *virtual* and can be overridden by the user-derived classes to implement user-defined functions.

**Important Note:** In order to use the user-defined classes based on this class, you must use the appropriate CIDocLink constructor. The appropriate CIDocLink constructor includes an argument that accepts a pointer to an instance of the user-derived class based on this class.

## CIDocOutboundFunc Class Summary

The CIDocOutboundFunc class is derived from CrfcServerFunc. You can use CIDocOutboundFunc as a base class for deriving user-defined classes and adding user-defined functionality. However, this class offers basic implementation that must be part of the execution of the user-derived class. Specifically, when an application program overrides the function Process() in a derived class, it must call CIDocOutboundFunc::Process() first. The basic implementation of Process() includes the following:

- After receiving IDocs from R/3, Process() fetches metadata about the IDOC\_DATA and IDOC\_CONTROL fields from R/3.
- Process() encapsulates each IDoc from the just-received batch of IDocs in a CIDoc object.

Of course, the user-derived class is entirely free to develop its own IDoc processing logic: you need not use any of the implementation in CIDocOutboundFunc::Process().

---

**CIDocTID Class Summary**

## CIDocTID Class Summary

This class provides functions for managing the transaction ID for transactional RFC. You can derive your own TID management class from this class and override the member functions with your own customized functions.

The construction of a CIDocTID object requires that the application program specify:

- Path and name of the TID management file.
- Path and name of the lock file.
- File extension for the file that will store the IDocs, in case of a failed call to the R/3 system.

The functions offered by this class are:

- Finding a TID in the TID management file.
- Updating the state of a specific TID in the TID management file.
- Writing a specific state for a TID into the TID management file.
- Searching for any TID whose call has failed.
- Locking the TID management file to prevent use by others.
- Unlocking the TID management file to enable use by others.

## **Basic Logic for Using the Class Library**

This section describes the basic structure of an application program when using the IDoc Class Library. It shows the sequences of object instantiations and function calls for creation, sending, receiving and processing of IDocs.

[Sending IDocs Inbound to R/3 \[Seite 30\]](#)

[Receiving IDocs Outbound from R/3 \[Seite 31\]](#)

---

**Sending IDocs Inbound to R/3**

## Sending IDocs Inbound to R/3

A program that sends IDocs inbound to R/3 is a transactional RFC client that creates IDocs from scratch and sends them to R/3 by making a tRFC call.

1. Instantiate an object of the CRfcConnection class.  
For information, see [RFC C++ Class Library \[Extern\]](#).
2. Set connection parameters (user and system parameters) in the object.
3. Instantiate a CIDocTID object for managing the transactional ID.
4. Instantiate a CIDocLink object.  
Specify this object to be an *inbound* link, along with the CRfcConnection object and the CIDocTID object.
5. For each IDoc needed:
  - a) Call AddIDoc (of the CIDocLink object) to add the IDoc into the link.  
Specify the IDoc type, R/3 release and the IDoc number. The AddIDoc function returns a reference to an object of the CIDoc class. Use this reference to access the IDoc just added.
  - b) For each IDoc segment needed:
    - i) Instantiate a CIDocIterator object for the Idoc:  

```
CIDocIterator it1(IDoc1);
```
    - ii) Use the AppendSegment function (of the CIDoc object) to create an empty segment.  

```
it1 = IDoc1.AppendSegment("E1MARAM");
```
    - iii) Use the Field member function (of the CIDocIterator object) to fill in field values for the segment.  

```
it1.Field("SPRAS") = "E";
```
6. Call SendIDocsToR3 (in the CIDocLink object) to send the created IDocs to R/3.

Use try-catch blocks throughout the program to catch exceptions thrown by the class library and to print out information about the exception.

## Receiving IDocs Outbound from R/3

A program that receives IDocs outbound from R/3 is implemented as a transactional RFC server that offers the functions "INBOUND\_IDOC\_PROCESS" and "IDOC\_INBOUND\_ASYNCHRONOUS". The server program mainly idles and waits for R/3 to call. When R/3 calls, the server program receives the IDocs and processes them, then returns to an idle state and continues to wait.

1. Instantiate an object of the CRfcConnection class.  
For information, see [RFC C++ Class Library \[Extern\]](#).
2. Set connection parameters (user and system parameters) in the object.
3. Instantiate a CIDocTID object for managing the transactional ID.
4. Instantiate a CIDocLink object.  
Specify it to be an *outbound* link, along with the CRfcConnection object and the CIDocTID object.
5. Loop to process outbound IDocs from R/3:
  - a) Call GetIDocsFromR3 (of the CIDocLink object) to wait for Idocs from R/3.
  - b) When IDocs are received, call GetIDocCount (of the CIDocLink object) to get the total number of IDocs sent.
  - c) Loop through the IDocs received.
    - i) Call GetIDoc (of the CIDocLink object) to access an IDoc.
    - ii) Instantiate a CIDocIterator object to access the segments of this IDoc.
    - iii) Traverse the segments of the IDoc using the CIDocIterator functions to:
      - position the iterator
      - access the fields in the segment pointed at by the iterator.

Use try-catch blocks all throughout the program to catch exceptions thrown by the class library and to print out information about the exception.

## Class Library Reference

This chapter describes the classes offered in the IDoc Class Library and their member functions.



In the function declarations to follow, 'string' is the string template class in the Standard C++ Library in Microsoft Visual C++ 5.0. The include file **<string>** must be included.

[The CIDocLink Class \[Seite 87\]](#)

[The CIDoc Class \[Seite 33\]](#)

[The CIDocIterator Class \[Seite 71\]](#)

[The CIDocFieldIterator Class \[Seite 45\]](#)

[The CIDocMetaDataTree Class \[Seite 103\]](#)

[The CIDocDataSegment Class \[Seite 120\]](#)

[The CIDocFieldInfo Class \[Seite 122\]](#)

[The CIDocTID Class \[Seite 123\]](#)

## The CIDoc Class

### Class Summary

The *CIDoc* class is defined in *idocidoc.h*.

An object of the *CIDoc* class encapsulates one and only one IDoc. The *CIDoc* class provides member functions for retrieving IDoc attributes or manipulating the IDoc by appending new segments to it. There is no public constructor for this class.

Note that you can use *CIDoc* member functions only through references to actual objects of the class. For example:

```
CIDoc& Idoc1 = IDocLink.AddIDoc(...);
```

### Operations

#### [AppendSegment \[Seite 34\]](#)

Appends a new segment with specified name to the IDoc.

#### [GetControlField \[Seite 35\]](#)

Returns a string object containing the value of the specified field in the control segment of the IDoc.

#### [GetIDocType \[Seite 37\]](#)

Returns the IDoc type for the encapsulated IDoc.

#### [GetIDocNumber \[Seite 36\]](#)

Returns the IDoc number for the encapsulated IDoc.

#### [GetRelease \[Seite 41\]](#)

Returns the R/3 release in which the IDoc type of the encapsulated IDoc was defined or upgraded.

#### [SetControlField \[Seite 42\]](#)

Sets the value of the specified field in the control segment of the IDoc.

#### [WriteIDocToFile \[Seite 44\]](#)

Saves the contents of the encapsulated IDoc to a file.

**AppendSegment**

## AppendSegment

### Purpose

Appends a new segment with the specified name to the IDoc.

### Syntax

```
CIDocIterator AppendSegment(string SegmentType);
```

### Parameters

*SegmentType*: a string object containing the type of the segment to be appended.

### Return Value

A CIDocIterator object that points to the newly appended segment.

### Exceptions

Throws IDOC\_ERROR\_INFO if a segment with the specified type cannot be placed at the end position of the IDoc.

### Description

This function appends a new, empty row to the IDoc only if a segment with the specified type can be legally placed there, given the current structure of the IDoc. For example, if the IDoc being constructed is of type MATMAS02, and if the last segment in the IDoc is of type E1MARDM, then the attempt to append a segment of type E1MAKTM would be illegal according to the definition of the IDoc type.

Important note: The argument must be a segment type, prefixed with „E1“ or „Z1“. If the argument is prefixed with "E2", "Z2" or anything else, this function throws an exception.

## GetControlField

### Purpose

Returns a string object containing the value of the specified field in the control segment of the IDoc.

### Syntax

```
string GetControlField(const string& strFieldName) const;
```

### Parameters

*strFieldName*: a `string` object containing the name of the desired field in the control segment.

### Return Value

A `string` object containing the value of the specified field.

### Exceptions

Throws `IDOC_ERROR_INFO` if the specified field is not found.

---

GetIDocNumber

## GetIDocNumber

### Purpose

Returns the IDoc number for the encapsulated IDoc.

### Syntax

```
string GetIDocNumber(void);
```

### Parameters

*None.*

### Return Value

A string object containing the IDoc number.

## GetIDocType

### Purpose

Returns the IDoc type for the encapsulated IDoc.

### Syntax

```
string GetIDocType (void) ;
```

### Parameters

*None.*

### Return Value

A string object containing the IDoc type.

**GetMessageType**

## GetMessageType

**Purpose**

Returns the message type for the IDoc.

**Syntax**

```
string GetMessageType() const;
```

**Parameters**

*None.*

**Return Value**

A `string` object containing the value of the field MESTYP in the control segment of the IDoc.

## GetPartner

### Purpose

Returns the name of the partner for the IDoc.

### Syntax

```
string GetPartner() const;
```

### Parameters

*None.*

### Return Value

A `string` object containing the value of the field SNDPRN in the control segment of the IDoc.

**GetPort**

## GetPort

### Purpose

Returns a name of the port for the IDoc.

### Syntax

```
string GetPort() const;
```

### Parameters

*None.*

### Return Value

A `string` object containing the value of the field SNDPRT in the control segment of the IDoc.

## GetRelease

### Purpose

Returns the R/3 release in which the IDoc type for this IDoc was defined or upgraded.

### Syntax

```
string GetRelease(void);
```

### Parameters

*None.*

### Return Value

A string object containing the release number.

---

**SetControlField**

## SetControlField

### Purpose

Sets the value of the specified field in the control segment of the IDoc.

### Syntax

```
void CIDoc::SetControlField(const string& strFieldName, const string& strValue)
```

### Parameters

- *strFieldName*: a `string` object containing the name of the desired field in the control segment.
- *strValue*: a `string` object containing the value to be set to the desired field in the control segment.

### Return Value

A `string` object containing the value of the specified field

### Exceptions

Throws `IDOC_ERROR_INFO` if the specified field is not found.

## SetIDocNumber

### Purpose

Stores the IDoc number for this IDoc.

### Syntax

```
void SetIDocNumber(const string& DocNumber);
```

### Parameters

*DocNumber*: a `string` object containing the IDoc number for this IDoc.

### Return Value

*None*.

---

**WriteIDocToFile**

## WriteIDocToFile

### Purpose

Saves the contents of the encapsulated IDoc to a file.

### Syntax

```
BOOL WriteIDocToFile(const string& filename);
```

### Parameters

*filename*: path and name of the file to which the IDoc contents are to be written.

### Return Value

A flag indicating whether the write operation was successful.

## The CIDocFieldIterator Class

### Class Summary

The *CIDocFieldIterator* class is defined in *idocitr.h*.

Use *CIDocFieldIterator* class objects are used for iterating through the fields or randomly access any field in an IDoc segment.

### Construction

#### [CIDocFieldIterator \(copy constructor\) \[Seite 47\]](#)

Constructs an object of CIDocFieldIterator class from another object of this class.

### Operations

#### [First \[Seite 48\]](#)

Positions the iterator on the first field of the segment.

#### [GetFieldInfo \[Seite 49\]](#)

Returns reference to CIDocFieldInfo object describing the current field

#### [GetFloatValue \[Seite 50\]](#)

Converts the value of the field to floating-point number and returns it.

#### [GetInternalType \[Seite 51\]](#)

Returns a string indicating the internal type of the current field.

#### [GetIntValue \[Seite 52\]](#)

Converts the value of the field to integer number and returns it.

#### [GetLength \[Seite 53\]](#)

Returns length of the current field in bytes.

#### [GetName \[Seite 54\]](#)

Returns name of current field.

#### [GetOffset \[Seite 55\]](#)

Returns offset of current field.

#### [GetStringValue \[Seite 56\]](#)

Returns the character string in the current field.

#### [IsEnd \[Seite 57\]](#)

Returns a flag indicating if the iterator is pointing to the last field.

#### [IsFirst \[Seite 58\]](#)

Returns a flag indicating if the iterator is pointing to the first field.

#### [Last \[Seite 59\]](#)

Positions the iterator on the last field of the segment.

**The CIDocFieldIterator Class**

[operator ++ \(pre-increment version\) \[Seite 60\]](#)

Advance the iterator to next field.

[operator ++ \(post-increment version\) \[Seite 61\]](#)

Advance the iterator to next field.

[operator -- \(pre-decrement version\) \[Seite 62\]](#)

Advance the iterator to previous field.

[operator -- \(post-decrement version\) \[Seite 63\]](#)

Advance the iterator to previous field.

[operator = \(floating-point\) \[Seite 64\]](#)

Assigns a floating-point value to the current field.

[operator = \(string\) \[Seite 65\]](#)

Assigns a character string value to the current field.

[operator = \(const char\\*\) \[Seite 66\]](#)

Assigns a character string value to the current field.

[operator = \(integer\) \[Seite 67\]](#)

Assigns an integer value to the current field.

[operator = \(assignment from another iterator\) \[Seite 68\]](#)

Copies another iterator's attributes.

[operator \[\] \(by index\) \[Seite 69\]](#)

Positions the iterator on the field specified by index.

[operator \[\] \(by name\) \[Seite 70\]](#)

Positions the iterator on the field specified by name of field.

## CIDocFieldIterator (copy constructor)

### Purpose

Constructs an object of CIDocFieldIterator class from another object of this class.

### Syntax

```
CIDocFieldIterator(const CIDocFieldIterator& ItSource);
```

### Parameters

*ItSource*: the source iterator object from which this object is to be copied and constructed.

### Return Value

None.

---

**First****First****Purpose**

Positions the iterator on the first field of the segment.

**Syntax**

```
CIDocFieldIterator& CIDocFieldIterator::First(void)
```

**Parameters**

*None.*

**Return Value**

Returns reference to the CIDocFielditerator object itself.

## GetFieldInfo

### Purpose

Returns a reference to a CIDocFieldInfo object describing the current field.

### Syntax

```
const CIDocFieldInfo& GetFieldInfo() const;
```

### Parameters

*None.*

### Return Value

Returns reference to CIDocFieldInfo object.

**GetFloatValue**

## GetFloatValue

### Purpose

Converts the value of the field to floating-point number and returns it.

### Syntax

```
float CIdocFieldIterator::GetFloatValue()
```

### Parameters

*None.*

### Return Value

Floating-point number representing the value of the field.

### Exceptions

Throws IDOC\_ERROR\_INFO for data conversion errors.

## GetInternalType

### Purpose

Returns a string indicating the internal type of the current field.

### Syntax

```
const string& CIDocFieldIterator::GetInternalType() const
```

### Parameters

*None.*

### Return Value

A string object containing the internal type of the field.

### Description

All fields in IDoc segments are of character type. However, the *internal type* of the field is the data type of the corresponding field in the R/3 application that generates the IDoc. The internal type can be any of the SAP data types (INT, INT2, CHAR, DEC, QUAN, etc.)

## GetIntValue

# GetIntValue

## Purpose

Converts the value of the field to integer number and returns it.

## Syntax

```
int CIDocFieldIterator::GetIntValue()
```

## Parameters

*None.*

## Return Value

Integer number representing the value of the field.

## Exceptions

Throws IDOC\_ERROR\_INFO for data conversion errors.

## GetLength

### Purpose

Returns length of the current field in bytes.

### Syntax

```
const int CIdocFieldIterator::GetLength() const
```

### Parameters

*None.*

### Return Value

Integer value of length.

---

GetName

## GetName

### Purpose

Returns name of current field.

### Syntax

```
const string& CIDocFieldIterator::GetName() const
```

### Parameters

*None.*

### Return Value

A string object containing the name of the field.

## GetOffset

### Purpose

Returns offset of current field.

### Syntax

```
const int CIdocFieldIterator::GetOffset() const
```

### Parameters

*None.*

### Return Value

Integer number representing the offset.

---

**GetStringValue**

## GetStringValue

### Purpose

Returns the character string in the current field.

### Syntax

```
string CIdocFieldIterator::GetStringValue()
```

### Parameters

*None.*

### Return Value

Integer number representing the value of the field.

### Exceptions

Throws IDOC\_ERROR\_INFO for data conversion errors.

## IsEnd

### Purpose

Returns a flag indicating if the iterator is pointing to the last field.

### Syntax

```
bool CIDocFieldIterator::IsEnd() const
```

### Parameters

*None.*

### Return Value

A `bool` value.

---

IsFirst

## IsFirst

### Purpose

Returns a flag indicating if the iterator is pointing to the first field.

### Syntax

```
bool CIdocFieldIterator::IsFirst() const
```

### Parameters

*None.*

### Return Value

A `bool` value.

## Last

### Purpose

Positions the iterator on the last field of the segment.

### Syntax

```
CIDocFieldIterator& CIDocFieldIterator::Last(void)
```

### Parameters

*None.*

### Return Value

Returns reference to the CIDocFielditerator object itself.

---

operator ++ (pre-increment version)

## operator ++ (pre-increment version)

### Purpose

Advance the iterator to next field.

### Syntax

```
CIDocFieldIterator& CIDocFieldIterator::operator++(void)
```

### Parameters

*None.*

### Return Value

Returns reference to the CIDocFielditerator object itself.

### Exceptions

Throws IDOC\_ERROR\_INFO if next field does not exist.

## operator ++ (post-increment version)

### Purpose

Advance the iterator to next field.

### Syntax

```
CIDocFieldIterator& CIDocFieldIterator::operator++(int i)
```

### Parameters

*None.*

### Return Value

Returns reference to the CIDocFielditerator object itself.

### Exceptions

Throws IDOC\_ERROR\_INFO if next field does not exist.

---

operator -- (pre-decrement version)

## operator -- (pre-decrement version)

### Purpose

Advance the iterator to previous field.

### Syntax

```
CIDocFieldIterator& CIDocFieldIterator::operator (void)
```

### Parameters

*None.*

### Return Value

Returns reference to the CIDocFielditerator object itself.

### Exceptions

Throws IDOC\_ERROR\_INFO if previous field does not exist.

## operator -- (post-decrement version)

### Purpose

Advance the iterator to previous field.

### Syntax

```
CIDocFieldIterator& CIDocFieldIterator::operator--(int i)
```

### Parameters

*None.*

### Return Value

Returns reference to the CIDocFielditerator object itself.

### Exceptions

Throws IDOC\_ERROR\_INFO if previous field does not exist.

operator = (floating-point)

## operator = (floating-point)

### Purpose

Assigns a floating-point value to the current field.

### Syntax

```
void CIDocFieldIterator::operator=(const float& dValue)
```

### Parameters

*dValue*: a floating-point number.

### Return Value

None.

### Exceptions

Throws IDOC\_ERROR\_INFO for data conversion errors.

## operator = (string)

### Purpose

Assigns a character string value to the current field.

### Syntax

```
void CIDocFieldIterator::operator=(const string& strValue)
```

### Parameters

*strValue*: a `string` object containing the input character string.

### Return Value

None.

### Exceptions

Throws `IDOC_ERROR_INFO` for data conversion errors.

---

`operator = (const char*)`

## **operator = (const char\*)**

### **Purpose**

Assigns a character string value to the current field.

### **Syntax**

```
void CIDocFieldIterator::operator=(CSTR rstrValue)
```

### **Parameters**

*strValue*: a character pointer pointing to the input character string.

### **Return Value**

None.

### **Exceptions**

Throws IDOC\_ERROR\_INFO for data conversion errors.

## operator = (integer)

### Purpose

Assigns an integer value to the current field.

### Syntax

```
void CIDocFieldIterator::operator=(const int nIntValue)
```

### Parameters

*nIntValue*: input integer number.

### Return Value

None.

### Exceptions

Throws IDOC\_ERROR\_INFO for data conversion errors.

---

`operator =` (assignment from another iterator)

## **operator = (assignment from another iterator)**

### **Purpose**

Copies another iterator's attributes.

### **Syntax**

```
CIDocFieldIterator& CIDocFieldIterator::operator=(const  
CIDocFieldIterator& ItSource)
```

### **Parameters**

*ItSource*: the *CIDocFieldIterator* object whose attributes are to be copied.

### **Return Value**

Returns reference to the *CIDocFieldIterator* object itself.

## operator [] (by index)

### Purpose

Positions the iterator on the field specified by index.

### Syntax

```
CIDocFieldIterator& CIDocFieldIterator::operator[] (const int nIndex)
```

### Parameters

*nIndex*: index of desired field.

### Return Value

Returns reference to the CIDocFielditerator object itself.

### Exceptions

Throws IDOC\_ERROR\_INFO if *nIndex* is too large

operator [] (by name)

## operator [] (by name)

### Purpose

Positions the iterator on the field specified by name of field.

### Syntax

```
CIDocFieldIterator& CIDocFieldIterator::operator[] (const string&  
strFieldName)
```

### Parameters

*strFieldName*: name of desired field.

### Return Value

Returns reference to the CIDocFielditerator object itself.

### Exceptions

Throws IDOC\_ERROR\_INFO if *strFieldName* is not found

## The CIDocIterator Class

### Class Summary

The *CIDocIterator* class is defined in *idocitr.h*.

Use *CIDocIterator* class objects to iterate through the segments of an IDoc. Each object of this class, referred to as an *iterator*, is associated with one and only one IDoc. However, you may associate more than one iterator with the same IDoc. Iterators can advance through an IDoc's hierarchical tree structure only in a forward and downward manner, except when they are reset to the beginning of the IDoc. Thus iterators may not be re-positioned on segments with lower sequence numbers than the current segment.

### Construction

[CIDocIterator \[Seite 73\]](#)

**Constructs an object of CIDocIterator class.**

### Operations

[Field \(write to field\) \[Seite 74\]](#)

Modifies the contents of the specified field.

[Field \(read from field\) \[Seite 75\]](#)

Reads the contents of the specified field.

[GoTop \[Seite 76\]](#)

Positions the iterator to the beginning of the IDoc.

[NextSibling \[Seite 77\]](#)

Advances the iterator to the next segment in the current segment group.

[NextRepetition \[Seite 78\]](#)

Advances the iterator to the next identically-named segment within the current segment group.

[FirstChild \[Seite 79\]](#)

Re-positions the iterator to the first child segment for the current parent.

[GetChild \[Seite 80\]](#)

Retrieves the child segment specified by an index.

[GetName \[Seite 81\]](#)

Returns the name of the segment on which the iterator is currently positioned.

[GetChildCount \[Seite 82\]](#)

Returns the total number of child segments for the parent.

[GetSegmentCount \[Seite 83\]](#)

Returns the total number of child segments with a specified name.

[GetFirstChildWithName \[Seite 84\]](#)

---

**The CIDocIterator Class**

Retrieves the first child segment with a specified name.

[operator ++ \[Seite 85\]](#)

Functions identically to the NextRepetition() member function.

[IsThereMoreOfThis \[Seite 86\]](#)

Queries the existence of other identically-named sibling segments.

## CIDocIterator

### Purpose

Constructs an object of CIDocIterator class.

### Syntax

```
CIDocIterator(CIDoc& IDoc) ;
```

### Parameters

*IDoc*: reference to a CIDoc object.

### Description

Use this function to construct an object of the CIDocIterator class. The parameter passed is a reference to the CIDoc within which this CIDocIterator will iterate. The CIDocIterator will iterate within that IDoc and that IDoc only.

---

Field (write to field)

## Field (write to field)

### Purpose

Modifies the contents of a given field.

### Syntax

```
CIDocData& Field(string fieldname);
```

### Parameters

*fieldname*: Name of the field to be accessed.

### Return Value

A reference to a CIDocData& object that can be used for data conversion purposes.

### Exceptions

An IDOC\_ERROR\_INFO structure is thrown for exceptions such as type mismatch and bad field name.

### Description

Use this function on the left-hand-side of an assignment statement to convert and store data from an application variable in the field. The iterator must be positioned on the segment containing the field.

## Field (read from field)

### Purpose

Reads the contents of a given field.

### Syntax

```
const CIDocData& Field(string fieldname) const;
```

### Parameters

*fieldname*: Name of the field to be accessed.

### Return Value

A reference to a const CIDocData object.

### Exceptions

An IDOC\_ERROR\_INFO structure is thrown for exceptions such as type mismatch and bad field name.

### Description

Use this function on the right-hand side of an assignment to retrieve the field's contents and assign them to a variable. The iterator must be positioned on the segment containing the field. The returned value is a constant (cannot be modified by the caller), but can be used to convert the data to the appropriate type.

---

GoTop

## GoTop

### Purpose

Positions the iterator to the beginning of the IDoc.

### Syntax

```
void GoTop(void);
```

### Parameters

*None.*

### Return Value

*None.*

## NextSibling

### Purpose

Advances the iterator to the next sibling segment in the current group.

### Syntax

```
BOOL NextSibling(void) ;
```

### Parameters

*None.*

### Return Value

A flag indicating whether the iterator has been re-positioned.

### Description

This function positions the iterator on the next sibling segment in the group, regardless of segment name. NextSibling returns FALSE if the iterator already on the last sibling in the current segment group. In this case, the iterator stays positioned where it was.

### Exceptions

*None.*

---

**NextRepetition**

## NextRepetition

### Purpose

Advances the iterator to the next identically-named sibling segment.

### Syntax

```
BOOL NextRepetition(void);
```

### Parameters

*None.*

### Return Value

A flag indicating whether the iterator has been re-positioned. If FALSE, the iterator stays where it was because it is already on the last segment with that name in the current segment group.

### Description

This function positions the iterator on the next identically-named sibling in the group. NextRepetition returns FALSE if the iterator already on the last sibling with the given name in the current group. In this case, the iterator stays positioned where it was.

### Exceptions

*None.*

## FirstChild

### Purpose

Re-positions the iterator to the first child segment in the segment group.

### Syntax

```
BOOL FirstChild(void);
```

### Parameters

*None.*

### Return Value

A flag indicating whether the iterator has been re-positioned. If FALSE, the iterator stays where it was because either there is no child or the iterator is not positioned on a parent segment.

### Description

This function positions the iterator on the first child segment for the current parent. The iterator must be positioned on a parent segment. This function returns FALSE if the iterator is not positioned on a parent segment. In this case, the iterator stays positioned where it was.

## GetChild

# GetChild

## Purpose

Retrieves the child segment specified by an index.

## Syntax

```
CIDocIterator GetChild(int nIndex);
```

## Parameters

*nIndex*: an integer that specifies a given child segment.

## Return Value

A CIDocIterator object positioned on the requested child segment.

## Description

This function retrieves the child segment specified by *nIndex*, and returns an iterator positioned to this child. The *nIndex* parameter is zero-based. The current iterator must be positioned on the relevant parent segment, and is not re-positioned by calling this function. You can assign the returned object to the iterator itself or another iterator.

## Exceptions

An IDOC\_ERROR\_INFO structure is thrown if the iterator is not positioned on a parent segment, or *nIndex* is larger than the total number of child segments in the segment group.

## GetName

### Purpose

Returns the name of the segment on which the iterator is currently positioned.

### Syntax

```
string GetName(void);
```

### Parameters

*None.*

### Return Value

A string object containing the name of the segment.

---

**GetChildCount**

## GetChildCount

### Purpose

Returns the total number of child segments for the current parent.

### Syntax

```
int GetChildCount(void);
```

### Parameters

*None.*

### Return Value

An integer containing the total number of child segments.

### Description

This function returns the total number of child segments for the segment on which the iterator is positioned. No grandchild or other descendant segments are counted. The current iterator must be positioned on the relevant parent segment.

### Exceptions

An IDOC\_ERROR\_INFO structure is thrown if the iterator is not positioned on a parent segment.

## GetSegmentCount

### Purpose

Returns the total number of child segments with a specified name.

### Syntax

```
int GetSegmentCount(string segname);
```

### Parameters

*segname*: name of the child segments to be counted.

### Return Value

A integer telling the total number of child segments with given name.

### Description

This function returns the total number of child segments with the specified name for a given parent segment. The iterator must be positioned on the relevant parent segment.

---

**GetFirstChildWithName**

## GetFirstChildWithName

### Purpose

Retrieves the first child segment with a specified name.

### Syntax

```
CIDocIterator GetFirstChildWithName(string segname);
```

### Parameters

*segname*: name of the first child segment.

### Return Value

A CIDocIterator object specifying the desired child segment.

### Description

This function retrieves the first child segment with a specified name. The current iterator must be positioned on the relevant parent segment, and is not re-positioned by calling this function. You can assign the returned object to the iterator itself or another iterator.

### Exceptions

An IDOC\_ERROR\_INFO structure is thrown if the iterator is not positioned on a parent segment.

## operator ++

### Purpose

Functions identically to the NextRepetition() member function.

### Syntax

```
BOOL operator++(void);
```

---

**IsThereMoreOfThis**

## IsThereMoreOfThis

### Purpose

Queries the existence of other identically-named sibling segments.

### Syntax

```
BOOL IsThereMoreOfThis (void) ;
```

### Parameters

*None.*

### Return Value

A flag indicating that other identically-named sibling segments exist.

### Description

This function searches in the list of sibling segments for other siblings with the same name as the current segment. The function only searches forward from the current position of the iterator. As a result, any sibling segments found will have sequence numbers higher than the current segment.

## The CIDocLink Class

### Class Summary

The *CIDocLink* class is defined in *idoclink.h*.

The *CIDocLink* class objects establishes links with R/3 to send and receive IDocs. In the case of an inbound link, a *CIDocLink* allows you to enter newly-created IDocs into the link for sending to R/3. In the case of an outbound link, *CIDocLink* member functions let you wait and receive IDocs from R/3. If a transactional RFC call fails while sending IDocs to R/3, the content of the IDocs is saved to a file for future attempts.

### Construction

[CIDocLink \[Seite 91\]](#)

**Constructs a CIDocLink object.**

### Operations

[GetIDocsFromR3 \[Seite 98\]](#)

Waits for R/3 to call in. When it does, receives the IDocs from R/3.

[SendIDocsToR3 \[Seite 102\]](#)

Sends created IDocs to R/3.

[GetIDocCount \[Seite 97\]](#)

Obtains the number of IDocs currently contained in the CIDocLink object.

[GetIDoc \[Seite 96\]](#)

Obtains a reference to the CIDoc object encapsulating the IDoc specified by the parameter.

[AddIDoc \(stored in file\) \[Seite 88\]](#)

Reads an IDoc from a file and adds it to the list of IDocs contained in this CIDocLink object.

[AddIDoc \(created from scratch\) \[Seite 89\]](#)

Creates an empty IDoc and adds it to the list of IDocs contained in this CIDocLink object.

[RemoveAllIDocs \[Seite 100\]](#)

Removes all IDocs contained in the CIDocLink object.

[ResendFailedIDocs \[Seite 101\]](#)

Re-sends IDocs associated with a Transaction ID that failed.

[CopyMetaDataTree \[Seite 93\]](#)

Returns a CIDocMetaDataTree object containing a copy of the metadata for a given IDoc type.

[GetTIDMgr \[Seite 99\]](#)

Returns the pointer to the CIDocTID object contained in this CIDocLink object.

---

AddIDoc (stored in file)

## AddIDoc (stored in file)

### Purpose

Gets an Idoc from a file and adds it to the list of IDocs contained in this CIDocLink object.

### Syntax

```
CIDoc& AddIDoc(const string& filename);
```

### Parameters

*filename*: Path and name of the file from which the content of the IDoc is to be read.

### Return Value

A reference to the CIDoc object that encapsulates the IDoc specified by the parameter.

### Exceptions

Throws const char\* for memory exceptions.

Throws IDOC\_ERROR\_INFO for all other exceptions.

### Description

This function is used for an inbound link when IDocs are created and added into the link for eventual transmission to R/3.

## AddIDoc (created from scratch)

### Purpose

Adds an empty IDoc to the list of IDocs contained in this CIDocLink object.

### Syntax

```
CIDoc& AddIDoc(const string& IDocType, const string& Release);
```

### Parameters

*IDocType*: Type of IDoc to be created.

*Release*: R/3 Release in which the IDoc type is defined or upgraded.

### Return Value

A reference to the CIDoc object created.

### Exceptions

Throws const char\* for memory exceptions.

Throws IDOC\_ERROR\_INFO for all other exceptions.

### Description

This function is used for an inbound link when IDocs are created and added into the link for eventual transmission to R/3. Note that after the call, the new IDoc is empty and needs to be filled segment by segment.

---

AddIDoc (from scratch, for extension types)

## AddIDoc (from scratch, for extension types)

### Purpose

Adds an empty IDoc to the list of IDocs contained in this CIDocLink object. This new IDoc is specified by its basic type and extension type.

### Syntax

```
CIDoc& AddIDoc(const string& IDocType,  
              const string& ExtendedType,  
              const string& Release);
```

### Parameters

*IDocType*: Basic type of IDoc to be created.

*ExtendedType*: Extended type of the IDoc.

*Release*: R/3 Release in which the IDoc type is defined or upgraded.

### Return Value

A reference to the CIDoc object created.

### Exceptions

Throws `const char*` for memory exceptions.

Throws `IDOC_ERROR_INFO` for all other exceptions.

### Description

This function is used for an inbound link when IDocs are created and added into the link for eventual transmission to R/3. Note that after the call, the new IDoc is empty and needs to be filled segment by segment.

## CIDocLink

### Purpose

Constructs a CIDocLink object.

### Syntax

```
CIDocLink(int LinkMode,  
          CRfcConnection *pConnection,  
          CIDocTID *pTidmgr,  
          INBOUND_METHOD inbound_method);
```

### Parameters

*LinkMode*: specifies the link mode: an INBOUND\_LINK or an OUTBOUND\_LINK.

*pConnection*: pointer to a CRfcConnection object with connection parameters.

*pTidmgr*: pointer to an object of the CIDocTID class or a class derived from it.

*Inbound\_method*: specifies the method with which the inbound IDocs are to be sent.

### Exceptions

Throws const char\* for memory exceptions.

Throws IDOC\_ERROR\_INFO for all other exceptions.

### Description

The link mode must be specified to indicate the desired direction.

The connection object's pointer is stored in the CIDocLink object for use when making or accepting calls to or from R/3. If no connection object is provided at construction time, the connection parameters must be set (using SetConnection) before making or accepting calls. Once set, these two parameters are used for all subsequent calls.

The argument *inbound\_method* indicates the preferred method for sending inbound IDocs. This is relevant only if the receiving R/3 is release 4.0 or greater. For earlier releases, this argument defaults to the value INBOUND\_METHOD\_3. See [R/3 Release-Dependent Issues \[Seite 14\]](#).

---

**CIDocLink (with user-defined outbound application)**

## CIDocLink (with user-defined outbound application)

### Purpose

Constructs a CIDocLink object, and imports an application into the CIDocLink object.

### Syntax

```
CIDocLink (CRfcConnection *pConnection,  
          CIDocOutboundApp *pOutApp,  
          CIDocTID *pTidmgr);
```

### Parameters

*pConnection*: pointer to a CRfcConnection object with connection parameters.

*pOutApp*: pointer to a user-defined outbound application object.

*pTidmgr*: pointer to an object of the CIDocTID class or a class derived from it.

### Description

With this constructor, you can define and implement your own outbound application and function objects and import them immediately into a CIDocLink object. The outbound application object should be an object of CIDocOutboundApp class or its derived class.

The newly created CIDocLink object uses the user-defined outbound application object in order to listen and wait for outbound IDocs. The ownership of the outbound application object always remains the user's, and it is the user's responsibility to destroy that object when it is no longer in use.

This constructor offers the flexibility to define and implement your own outbound application using the CIDocOutboundApp class or its derived class.

The connection object's pointer is stored in the CIDocLink object for use when making or accepting calls to (or from) R/3. If no connection object is provided at construction time, the connection parameters must be set (using SetConnection) before making or accepting calls. Once set, these two parameters are used for all subsequent calls.

## CopyMetaDataTree

### Purpose

Returns a CIDocMetaDataTree object containing a copy of the metadata for a given IDoc type.

### Syntax

```
CIDocMetaDataTree* CopyMetadataTree(const string& strIDocType,  
const string& strRelease)
```

### Parameters

*strIDocType*: a `string` object containing the type of the IDoc.

*strRelease*: a `string` object containing the release version of the IDoc type.

### Return Value

A pointer to a CIDocMetaDataTree containing metadata for the desired IDoc type. Returns NULL if the metadata of the desired IDoc type does not exist within this CIDocLink object

---

**ExitWaitLoop**

## ExitWaitLoop

### Purpose

Causes the process to exit from the infinite loop in CIDocLink::GetIDocsFromR3().

### Syntax

```
void ExitWaitLoop(void);
```

### Parameters

*None.*

### Return Value

*None.*

### Description

When the function CIDocLink::GetIDocsFromR3() executes, it loops infinitely, waiting for a remote function call from R/3 to pass IDocs to it. Should it be necessary to terminate this loop at any time, you may use this function to do so.

## GetConnection

### Purpose

Returns the CRfcConnection object used during the construction of this CIDocLink object.

### Syntax

```
CRfcConnection* GetConnection(void);
```

### Parameters

*None.*

### Return Value

The stored pointer to the CRfcConnection object.

### Description

When a CIDocLink object is constructed, a pointer to a CRfcConnection object is passed as an argument to the constructor and is stored. This function returns the stored pointer.

---

**GetIDoc**

## GetIDoc

### Purpose

Gets an CIDoc object from a list of CIDocs.

### Syntax

```
CIDoc& GetIDoc (int nIndex);
```

### Parameters

*nIndex*: specifies the index of the desired IDoc in the list of IDocs.

### Return Value

A reference to an CIDoc object.

### Exceptions

Throws IDOC\_ERROR\_INFO if nIndex exceeds the number of IDocs in the list.

### Description

This function returns reference to the CIDoc object specified by the nIndex parameter.

## GetIDocCount

### Purpose

Returns the number of IDocs currently contained in the CIDocLink object.

### Syntax

```
int GetIDocCount (void);
```

### Parameters

*None.*

### Return Value

An integer value telling the number of IDocs.

### Description

GetIDocCount returns the number of IDocs currently contained in the CIDocLink object. Use this function after receiving a group of IDocs from R/3.

---

**GetIDocsFromR3**

## GetIDocsFromR3

### Purpose

Waits for and receives the IDocs from R/3.

### Syntax

```
RFC_RC GetIDocsFromR3 (void) ;
```

### Parameters

*None*

### Return Value

RFC\_RC is the RFC return code defined in saprfc.h. See the [RFC API documentation \[Extern\]](#).

### Exceptions

Throws const char\* for memory exceptions.

Throws IDOC\_ERROR\_INFO for all other exceptions.

### Description

This function waits for the next group of IDocs coming in from R/3.

## GetTIDMgr

### Purpose

Returns the pointer to the CIDocTID object contained in this CIDocLink object.

### Syntax

```
Const CIDocTID* CIDocLink::GetTIDMgr() const
```

### Parameters

*None.*

### Return Value

A pointer to the CIDocTID object contained in this CIDocLink object.

---

**RemoveAllIDocs**

## RemoveAllIDocs

### Purpose

Removes all IDocs contained in the CIDocLink object.

### Syntax

```
void CIDocLink::RemoveAllIDocs(void)
```

### Parameters

*None.*

### Return Value

*None.*

## ResendFailedIDocs

### Purpose

Re-sends IDocs associated with a Transaction ID that failed.

### Syntax

```
void CIDocLink::ResendFailedIDocs(const RFC_TID szRfcTid)
```

### Parameters

*szRfcTid*: an RFC\_TID object containing the Transaction ID of the failed inbound call.

### Return Value

None.

### Exceptions

Throws IDOC\_ERROR\_INFO if the inbound call fails.

### Description

This function accepts the Transaction ID of a failed inbound call and uses that transaction ID, in combination with the file extension saved in the CIDocTID object, to access the file that contains the saved IDocs associated with the failed Transaction ID. These IDocs are loaded from the file and another transmission is attempted.

---

**SendIDocsToR3**

## SendIDocsToR3

### Purpose

Sends created IDocs to R/3.

### Syntax

```
void SendIDocsToR3 ();
```

### Parameters

None.

### Exceptions

Throws const char\* for memory exceptions.

Throws IDOC\_ERROR\_INFO for all other exceptions.

### Description

This function sends a group of IDocs to R/3.

## The CIDocMetaDataTree Class

### Class Summary

The *CIDocMetaDataTree* class is defined in *idocmeta.h*.

The *CIDocMetaDataTree* class encapsulates the tree structure of the metadata for a given IDoc type. The tree structure consists of an array of *CIDocDataSegment* objects that describe the data segments types that belong to the given IDoc type. Then, each *CIDocDataSegment* object contains a list of types of data segments that are its child segments. At last, each *CIDocDataSegment* object contains a list of *CIDocFieldInfo* class objects that describe each of its fields.

### Construction

[CIDocMetaDataTree \(copy constructor\) \[Seite 104\]](#)

Constructs an object of *CIDocMetaDataTree* class from another object of this class.

### Operations

[FindChildSegment \[Seite 105\]](#)

Returns a reference to a *CIDocDataSegment* object that describes the desired child segment.

[FindField \[Seite 106\]](#)

Returns reference to *CIDocFieldInfo* object describing a field, specified by the type or name of its parent segment and name of the field).

[FindSegment \[Seite 107\]](#)

Returns reference to *CIDocDataSegment* object (specified by type or name of the segment) describing a data segment.

[GetSegmentCount \[Seite 108\]](#)

Returns number of *CIDocDataSegment* objects contained in the metadata tree.

[operator \[\] \[Seite 109\]](#)

Positions the iterator on the field specified by name of field.

---

CIDocMetaDataTree (copy constructor)

## CIDocMetaDataTree (copy constructor)

### Purpose

Constructs an object of CIDocMetaDataTree class from another object of this class.

### Syntax

```
CIDocMetaDataTree::CIDocMetaDataTree (vector<CIDocDataSegment>&
source)
```

### Parameters

*source*: the source CIDocMetaDataTree object whose content is to be copied.

### Return Value

None.

## FindChildSegment

### Purpose

Returns the `CIDocDataSegment` object for the desired child segment.

### Syntax

```
const CIDocDataSegment& FindChildSegment(const CIDocDataSegment& seg, const  
int nIndex) const;
```

### Parameters

*seg*: a reference to the `CIDocDataSegment` object describing the *parent* segment of the requested child segment.

*nIndex*: an `integer` indicating the index of the desired child segment.

### Return Value

Returns a reference to the `CIDocDataSegment` object for the desired child segment.

### Exception

This function throws an `IDOC_ERROR_INFO` structure if metadata for the desired child segment is not found.

---

**FindField**

## FindField

### Purpose

Returns a given CIDocFieldInfo object in the tree.

### Syntax

```
const CIDocFieldInfo& FindField(const string& strSegment, const  
string& strField) const;
```

### Parameters

*strSegment*: a `string` object containing the type or name of the data segment whose field description is desired.

*strField*: a `string` object containing the name of the field whose description is desired.

### Return Value

Returns a reference to the CIDocFieldInfo object describing the field specified input parameters.

### Exceptions

Throws `IDOC_ERROR_INFO` if the desired CIDocFieldInfo is not found.

## FindSegment

### Purpose

Returns a given CIDocDataSegment object in the tree.

### Syntax

```
const CIDocDataSegment& FindSegment(const string&  
strSegmentTypeOrName) const;
```

### Parameters

*strSegmentType*: a *string* object containing the type or name of the desired data segment.

### Return Value

Returns a reference to the CIDocDataSegment object describing a data segment.

### Exceptions

Throws IDOC\_ERROR\_INFO if the desired CIDocDataSegment is not found.

---

**GetSegmentCount**

## GetSegmentCount

### Purpose

Returns number of CIDocDataSegment objects contained in the metadata tree.

### Syntax

```
int CIDocMetaDataTree::GetSegmentCount(void) const
```

### Parameters

*None.*

### Return Value

Returns integer number representing the count.

## operator []

### Purpose

Positions the iterator on the field specified by name of field.

### Syntax

```
const CIDocDataSegment& CIDocMetaDataTree::operator[] (int nIndex)  
const
```

### Parameters

*nIndex*: index of the CIDocDataSegment object in tree.

### Return Value

Returns reference to the desired CIDocDataSegment object in the tree.

### Exceptions

Throws IDOC\_ERROR\_INFO if *nIndex* is too large.

---

**The CIDocOutboundApp Class**

## The CIDocOutboundApp Class

### Class Summary

The *CIDocOutboundApp* class is defined in *idoclink.h*.

The *CIDocOutboundApp* class is derived from `CRfcTransApp` class and encapsulates the transactional server functionality for receiving IDocs from R/3. This class is intended as a base class for deriving classes that implement user-defined functionality. However, because this class provides simple implementations of the `OnCheckID()`, `OnCommit()`, `OnConfirmID()` and `OnRollBack()` functions, you can also use *CIDocOutboundApp* "as is" for quick prototyping.

**Note:** If you intend to use your own *CIDocOutboundFunc* objects or to derive and implement your own class based on this class, you should call the appropriate *CIDocLink* constructor.

### Construction

#### [CIDocOutboundApp \[Seite 111\]](#)

Constructs a *CIDocOutboundApp* object.

### Operations

#### [OnCheckID \[Seite 112\]](#)

Implements the `CRfcTransApp::OnCheckID()` function.

#### [OnCommit \[Seite 113\]](#)

Implements the `CRfcTransApp::OnCommit()` function.

#### [OnConfirmID \[Seite 114\]](#)

Implements the `CRfcTransApp::OnConfirm()` function.

#### [OnIdle \[Seite 115\]](#)

Implements the `CRfcTransApp::OnIdle()` function and sleeps 1 second.

#### [OnRollBack \[Seite 116\]](#)

Implements the `CRfcTransApp::OnRollBack()` function.

### See Also

[CIDocLink \(with user-defined outbound application\) \[Seite 92\]](#)

[The CIDocOutboundFunc Class \[Seite 117\]](#)

## CIDocOutboundApp

### Purpose

Constructs a CIDocOutboundApp object.

### Syntax

```
CIDocOutboundApp (CIDocTID *pTidmgr, CRfcConnection*  
pConnection) ;
```

### Parameters

*pTidmgr*: a pointer to a CIDocTID object.

*pConnection*: a pointer to a CRfcConnection object.

### Exceptions

Throws `const char*` if a calling program attempts to create more than one CRfcTransApp object.

## OnCheckID

# OnCheckID

## Purpose

Implements the CRfcTransApp::OnCheckID() function.

## Syntax

```
virtual int OnCheckID(RFC_TID tid);
```

## Parameters

*tid*: Transaction ID of the initiated transaction.

## Return Value

Returns an integer with value 0 if the function executed successfully. Returns the value 1 if one of the following occurred:

- The *tid* already exists in the default TID management file.
- An attempt to write to the file failed.

## Description

This function is called by the RFC Library when a transactional RFC server call is initiated.

This function first writes to a trace file (if enabled) to record the fact that this function was called. Then it calls CIdocTID::CheckTID() to write *tid* to the default TID management file. (See [CheckTID \[Seite 125\]](#).)

This function is declared *virtual*, and may therefore be overridden by a derived class. The implementation offered by this function is the default implementation, and is provided simply for the convenience of the user.

## OnCommit

### Purpose

Implements the CRfcTransApp::OnCommit() function.

### Syntax

```
virtual void OnCommit(RFC_TID tid);
```

### Parameters

*tid*: Transaction ID of the local transaction to be committed.

### Return Value

None.

### Description

This function is called by RFC Library when a transactional RFC server call is to be committed.

This function first writes to a trace file (if enabled) to record the fact that this function was called. Then it calls CIdocTID::UpdateTID() to change the state of *tid* to EXECUTED in the default TID management file. (See [UpdateTID \[Seite 126\]](#).)

If an error condition occurs during writing to the default TID management file, the trace file is updated with the relevant information.

This function is declared *virtual*, and may therefore be overridden by a derived class. The implementation offered by this function is the default implementation, and is provided simply for the convenience of the user.

## OnConfirmID

# OnConfirmID

## Purpose

Implements the CRfcTransApp::OnConfirm() function.

## Syntax

```
virtual void OnConfirmID(RFC_TID tid);
```

## Parameters

*tid*: Transaction ID of the local transaction to be confirmed.

## Return Value

*None*.

## Description

This function is called by RFC Library when a transactional RFC server call is to be confirmed.

This function first writes to a trace file (if enabled) to record the fact that this function was called. Then it calls CIdocTID::UpdateTID() to change the state of *tid* to CONFIRMED in the default TID management file. (See [UpdateTID \[Seite 126\]](#).)

If an error condition occurs during writing to the default TID management file, the trace file is updated with the relevant information.

This function is declared *virtual*, and may therefore be overridden by a derived class. The implementation offered by this function is the default implementation, and is provided simply for the convenience of the user.

## OnIdle

### Purpose

Implements the CRfcTransApp::OnIdle() function and sleeps for one second.

### Syntax

```
virtual void OnIdle();
```

### Parameters

*None.*

### Return Value

*None.*

### Description

This function is called by CRfcTransApp::Run(). The implementation provided here calls Sleep() for one second, in order to allow other processes on the host computer to run.

This function is declared *virtual*, and may therefore be overridden by a derived class. The implementation offered by this function is the default implementation, and is provided simply for the convenience of the user.

## OnRollBack

# OnRollBack

## Purpose

Implements the CRfcTransApp::OnRollBack() function.

## Syntax

```
virtual void OnRollBack(RFC_TID tid);
```

## Parameters

*tid*: Transaction ID of the local transaction to be rolled back.

## Return Value

None.

## Description

This function is called by RFC Library when a transactional RFC server call is to be rolled back.

This function first writes to a trace file (if enabled) to record the fact that this function was called. Then it calls CIdocTID::UpdateTID() to change the state of *tid* to ROLLBACK in the default TID management file. (See [UpdateTID \[Seite 126\]](#).)

When an error condition occurs during writing to the default TID management file, the trace file is updated with the relevant information.

This function is declared *virtual*, and may therefore be overridden by a derived class. The implementation offered by this function is the default implementation, and is provided simply for the convenience of the user.

## The CIDocOutboundFunc Class

### Class Summary

The *CIDocOutboundFunc* class is defined in *idoclink.h*.

The *CIDocOutboundFunc* class is derived from the *CRfcServerFunc* class and encapsulates the server functionality for executing on a desktop computer and processing IDocs received from an R/3 system. This class is intended for use as a base class for deriving a class for user-defined outbound processing functions. *CIDocOutboundFunc* achieves this by constructing the necessary table parameters for receiving IDocs and obtaining metadata for the fields in the table parameters from R/3., for ease of accessing the contents of the fields.

**Note:** In order to be used, objects of this class must be registered with an object of *CIDocOutboundApp* using the *CIDocOutboundApp::AddServerFunction()* function.

### Construction

[CIDocOutboundFunc \[Seite 118\]](#)

Constructs a *CIDocOutboundFunc* object.

### Operations

[Process \[Seite 119\]](#)

Processes the call from R/3 that transferred IDocs from the R/3 system to this application.

---

**CIDocOutboundFunc**

## CIDocOutboundFunc

### Purpose

Constructs a CIDocOutboundFunc object.

### Syntax

```
CIDocOutboundFunc(CIDocLink *pLink, const string& strFuncName);
```

### Parameters

*pLink*: a pointer to the parent CIDocLink object.

*strFuncName*: a `string` object containing the name of the server RFC function implemented by this CIDocOutboundFunc object.

### Exceptions

Throws `const char*` for memory exceptions.

## Process

### Purpose

Processes the call from R/3 that transferred IDocs from the R/3 system to this application.

### Syntax

```
virtual void Process();
```

### Parameters

*None.*

### Descriptions

The Process() function is declared *virtual* and is therefore intended to be overridden by a user implementation in a derived class.

The implementation provided for this function offers some basic (and necessary) functionality. Therefore, user-implemented functions that override the default implementation should call this function first, before continuing to execute user-defined functionality.

The basic functionality provided in this implementation includes:

1. Calling back to R/3, after receiving IDocs, to fetch the metadata for the table parameters IDOC\_DATA and IDOC\_CONTROL. The metadata allows you to access the table parameter fields more easily.
2. Extracting individual IDocs from the just-received batch of IDocs, and placing them in individual CIDoc objects.

---

**The CIDocDataSegment Class**

## The CIDocDataSegment Class

### Class Summary

The *CIDocDataSegment* class is defined in *idocdseg.h*.

Use *CIDocDataSegment* class objects encapsulate the metadata for an IDoc data segment type.

### Construction

The application program has no need for a constructor of this class.

### Operations

The following operations are available, but not further documented.

#### GetFieldCount

Returns the number of fields within this segment.

#### GetFieldsList

Returns a pointer to the list of CIDocFieldInfo objects contained in this segment

#### GetGrpMaxRep

Returns the number of maximum allowed repetitions for the IDoc segment group if this segment is a parent segment.

#### GetGrpMinRep

Returns the number of minimum allowed repetitions for the IDoc segment group if this segment is a parent segment.

#### GetGrpMustFlag

Returns a flag indicating whether the IDoc segment group is mandatory if this segment is a parent segment.

#### GetHierLevel

Returns hierarchical level of this segment.

#### GetLength

Returns length in bytes of this segment.

#### GetMaxRepetition

Returns maximum allowed number of repetitions.

#### GetMinRepetition

Returns minimum allowed number of repetitions.

#### GetMustFlag

Returns a flag indicating if the segment is mandatory.

#### GetName

Returns name of the segment.

#### GetParentName

Returns name of parent segment.

#### GetParentType

Returns type of parent segment.

---

The CIDocDataSegment Class

**GetType**

Returns type of this segment.

**IsParent**

Returns a flag indicating whether this segment is a parent segment.

**operator []**

Returns reference to the CIDocFieldInfo object (specified by its index) that describes a field.

---

## The CIDocFieldInfo Class

# The CIDocFieldInfo Class

## Class Summary

The *CIDocFieldInfo* class is defined in *idocdseg.h*.

Use *CIDocFieldInfo* class objects to encapsulate the metadata for an IDoc field.

## Construction

The application program has no need for a constructor of this class.

## Operations

The following operations are available, but not further documented.

### **GetInternalType**

Returns the internal type of this field.

### **GetLength**

Returns length of this field

### **GetName**

Returns name of this field.

### **GetOffset**

Returns offset of this field.

### **GetType**

Returns type of this field.

## The CIDocTID Class

### Class Summary

The *CIDocTID* class is defined in *idoctidm.h*.

The *CIDocTID* class provides functions for managing the transaction IDs (“TIDs”) in a file. You can derive your own TID management class from this class if you want to implement customized functions for managing TIDs.

### Construction

[CIDocTID \[Seite 124\]](#)

Constructs a CIDocTID object.

### Operations

[CheckTID \[Seite 125\]](#)

Locates (or appends) a given TID in the TID management file and retrieves its state.

[UpdateTID \[Seite 126\]](#)

Updates the state of a given TID in the TID management file.

[FindFailedTID \[Seite 127\]](#)

Locates the TID for a failed call in the TID management file.

[WriteTID \[Seite 128\]](#)

Writes a TID and its state to the TID management file.

[LockTID \[Seite 129\]](#)

Locks the TID management file to prevent use by others.

[UnlockTID \[Seite 130\]](#)

Unlocks the TID management file to enable use by others.

## CIDocTID

## CIDocTID

### Purpose

Constructs a CIDocTID object.

### Syntax

```
CIDocTID(char* TidFileName,  
         char* LockFileName,  
         char* FileExtension);
```

### Parameters

- *TidFileName*: path and name of the TID management file.
- *LockFileName*: path and name of the file for executing the TID lock.
- *FileExtension*: file extension for the file in which IDocs should be stored after a failed call.

## CheckTID

### Purpose

Locates (or appends) a given TID in the TID management file and retrieves the TID's state.

### Syntax

```
virtual TID_RC CheckTID(RFC_TID tid);
```

### Parameters

*tid*: the TID to be found.

### Description

This function searches the TID management file for the TID specified by the input parameter and retrieves the TID's state. If the TID does not exist in the management file, CheckTID appends it. If the TID management file itself does not exist, then CheckTID creates the file and writes the TID to it.

### Return Value

A TID\_RC value indicating the result.

---

**UpdateTID**

## UpdateTID

### Purpose

Updates the state of a given TID in the TID management file.

### Syntax

```
virtual TID_RC UpdateTID(RFC_TID tid, TID_STATE tid_state);
```

### Parameters

- *tid*: the TID to be updated.
- *tid\_state*: new state to be written to TID management file.

### Return Value

A TID\_RC value indicating the result.

## FindFailedTID

### Purpose

Finds the first TID for a failed call in the TID management file.

### Syntax

```
virtual TID_RC FindFailedTID(RFC_TID tid);
```

### Parameters

*tid*: contains the first TID found for a failed call, if any failed-call TIDs exist.

### Return Value

Returns TID\_FOUND if a failed TID is found, or TID\_OK if none is found.

### Description

This function searches the TID management file for any TIDs belonging to calls that failed. If a failed call is found, the TID is returned to the caller in the *tid* argument.

---

**WriteTID**

## WriteTID

### Purpose

Writes a TID and its state to the TID management file.

### Syntax

```
virtual int WriteTID(RFC_TID tid, TID_STATE tid_state);
```

### Parameters

- *tid*: TID to be written to the file.
- *tid\_state*: state of the TID.

### Return Value

Returns 1 if writing to the file failed.

Returns 0 if successful.

## LockTID

### Purpose

Locks the TID management file to prevent use by others.

### Syntax

```
virtual int LockTID(RFC_TID tid);
```

### Parameters

*tid*: TID for which the TID management file is locked.

### Return Value

Returns 1 if locking failed.

Returns 0 if successful.

### Description

The LockTID function locks the TID management file to prevent use by others. This function uses the TID passed in to identify the user locking the file.

---

**UnlockTID**

## UnlockTID

### Purpose

Unlocks the TID management file to enable use by others.

### Syntax

```
virtual int UnlockTID(void);
```

### Parameters

*None.*

### Return Value

Returns 1 if unlocking failed.

Returns 0 if successful.

## The CIDocTrace Class

### Class Summary

The *CIDocTrace* class is defined in *idocglob.h*.

The *CIDocTrace* class offers functiond for writing error information to a trace file. The implementation for this class allows only one instance of this class to be instantiated. As a result, CIDocTrace is not suitable for multi-threaded applications.

The trace files opened by this class have names of the form **idocnnnn\_nnnnn.trc**, where each *n* represents a numeric digit from '0' to '9' inclusive.

### Construction

Construction of objects for this class by the user is not allowed.

### Operations

#### [SetTrace \[Seite 132\]](#)

Turns tracing on or off.

#### [Trace \[Seite 133\]](#)

Writes text to a trace file (if tracing was previously enabled).

#### [TraceBlankLines \[Seite 134\]](#)

Writes blank lines to a trace file (if tracing was previously enabled).

#### [TraceIDOCError \[Seite 135\]](#)

Writes the content of an `IDOC_ERROR_INFO` structure to a trace file (if tracing was previously enabled.)

#### [TraceRFCError \[Seite 136\]](#)

Writes the content of an `RFC_ERROR_INFO` structure to a trace file (if tracing was previously enabled.)

#### [TraceTime \[Seite 137\]](#)

Writes the current date and time to a trace file (if tracing was previously enabled.)

---

**SetTrace**

## SetTrace

### Purpose

Turns tracing on or off.

### Syntax

```
void SetTrace(bool TraceOn) ;
```

### Parameters

*TraceOn*: a boolean flag that enables or disables tracing.

### Descriptions

To turn tracing on, set the **TraceOn** argument to TRUE. When you do, SetTrace first opens a default trace file (if it is not yet opened). All subsequent calls to the trace functions in this class cause text to be written to the default text file. Setting the **TraceOn** argument to FALSE blocks all subsequent trace calls from writing to the trace file.

## Trace

### Purpose

Writes text to the trace file (if tracing previously enabled).

### Syntax

```
void Trace(const string& strTraceMsg);
```

### Parameters

*strTraceMsg*: message to be written to trace file.

### Descriptions

This function writes the message text to the trace file only if tracing was previously enabled by a call to SetTrace().

---

**TraceBlankLines**

## TraceBlankLines

### Purpose

Writes blank lines to the trace file (if tracing previously enabled).

### Syntax

```
void TraceBlankLines(const int& nLineCount);
```

### Parameters

*nLineCount*: number of blank lines to be output to the trace file.

### Descriptions

This function writes blank lines to the trace file only if tracing was previously enabled by a call to SetTrace().

## TraceIDocError

### Purpose

Writes the contents of an `IDOC_ERROR_INFO` structure to the trace file (if tracing previously enabled).

### Syntax

```
void TraceIDocError(const IDOC_ERROR_INFO& err);
```

### Parameters

*err*: reference to an `IDOC_ERROR_INFO` structure whose contents are to be written to the trace file.

### Descriptions

This function writes the contents of the `IDOC_ERROR_INFO` structure to the trace file only if tracing was previously enabled by a call to `SetTrace()`.

---

**TraceRFCError**

## TraceRFCError

### Purpose

Writes the contents of an `RFC_ERROR_INFO` structure to the trace file (if tracing previously enabled).

### Syntax

```
void TraceRFCError(const RFC_ERROR_INFO& err);
```

### Parameters

*err*: reference to an `RFC_ERROR_INFO` structure whose contents are to be written to the trace file.

### Descriptions

This function writes the contents of the `RFC_ERROR_INFO` structure to the trace file only if tracing was previously enabled by a call to `SetTrace()`.

## TraceTime

### Purpose

Writes the current date and time to the trace file (if tracing previously enabled).

### Syntax

```
void TraceTime();
```

### Parameters

*None.*

### Descriptions

This function writes the current date and time to the trace file only if tracing was previously enabled by a call to SetTrace().