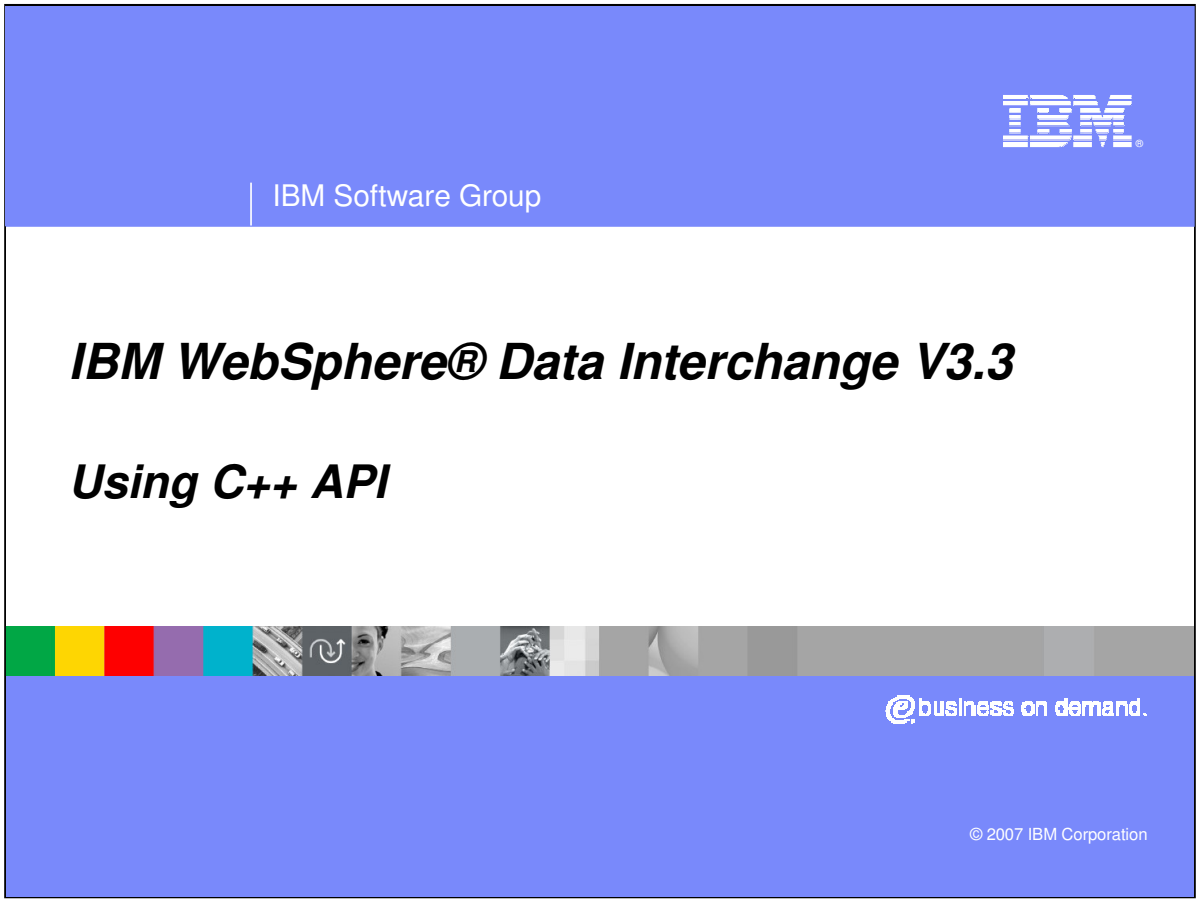


This presentation discusses the use of a C++ Application Program Interface to access the functions and features of WebSphere Data Interchange version 3.3.

Agenda

- About the C++ API
- API Architecture
- API in action
- Building an Application
- Summary

This discussion starts with a high level overview of the API. It then illustrates the use of the API in the sample program distributed with WebSphere Data Interchange on Windows and AIX. The discussion concludes with basic information about building an application that uses this API.

About the C++ API

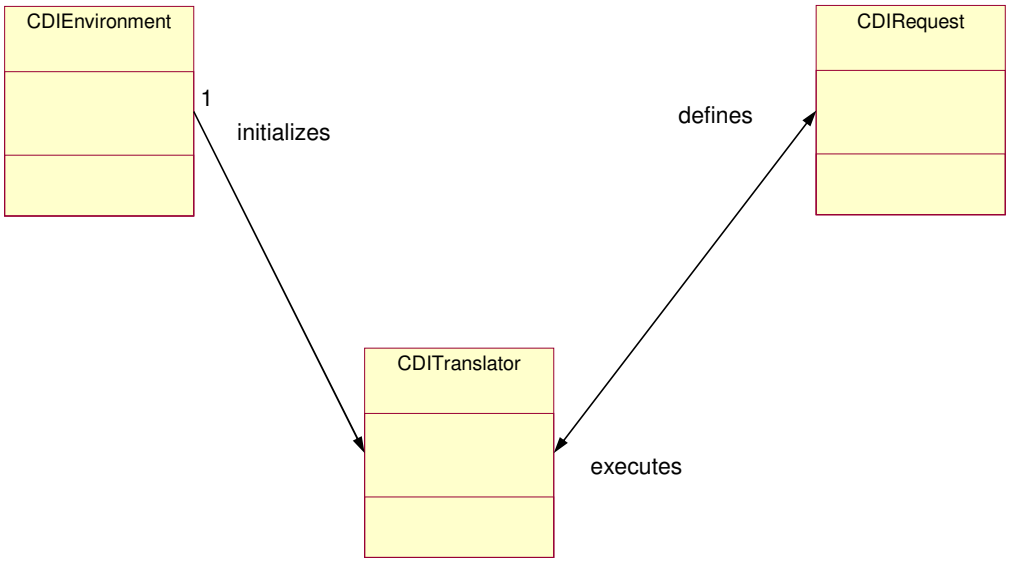
- Provides simple and efficient access to all features
- Supports both the command interface (ediservr) and the WebSphere MQ adapters
- Described in Programmer's Reference Guide : "Calling from a C++ program"
- Illustrated in ../samples/apiexamp.cpp

Using the C++ API, you can access the full range of translation and reporting features available in WebSphere Data Interchange . This interface handles many of the cumbersome details of manipulating the structures and control blocks that are used to communicate with the product's executable code. Even so, the C++ wrapper resolves quickly into product calls and does not add significant overhead to a client application.

This interface is used by the product to implement the "ediservr" command scripting interface that is the basis for execution on open platforms. It also supports both the triggered WebSphere MQ adapter and the WebSphere MQ adapter server.

Additional details can be found in WebSphere® Data Interchange for MultiPlatforms version 3.3 Programmer's Reference Guide, SC32-6217-01. A functional example of a client application performing an XML to EDI translation is furnished in the product's "samples" folder as "apiexamp.cpp." Program code later in this presentation come directly from this sample.

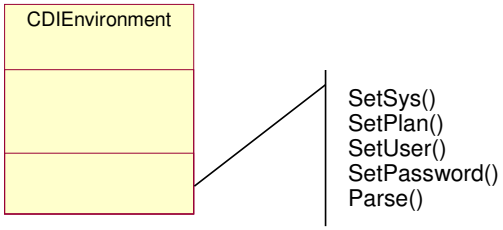
Principal classes



There are three classes that define the C++ interface: CDIEnvironment, CDITranslator and CDIRequest. An object of all three classes must be instantiated to perform translation.

An environment object tells a translator object how to interact with its runtime configuration. It is used only to initialize a translator object. A request object defines the action that a translator should perform. Many request objects can be handled by a single translator instance. A request object may be reused to perform the same or a different action.

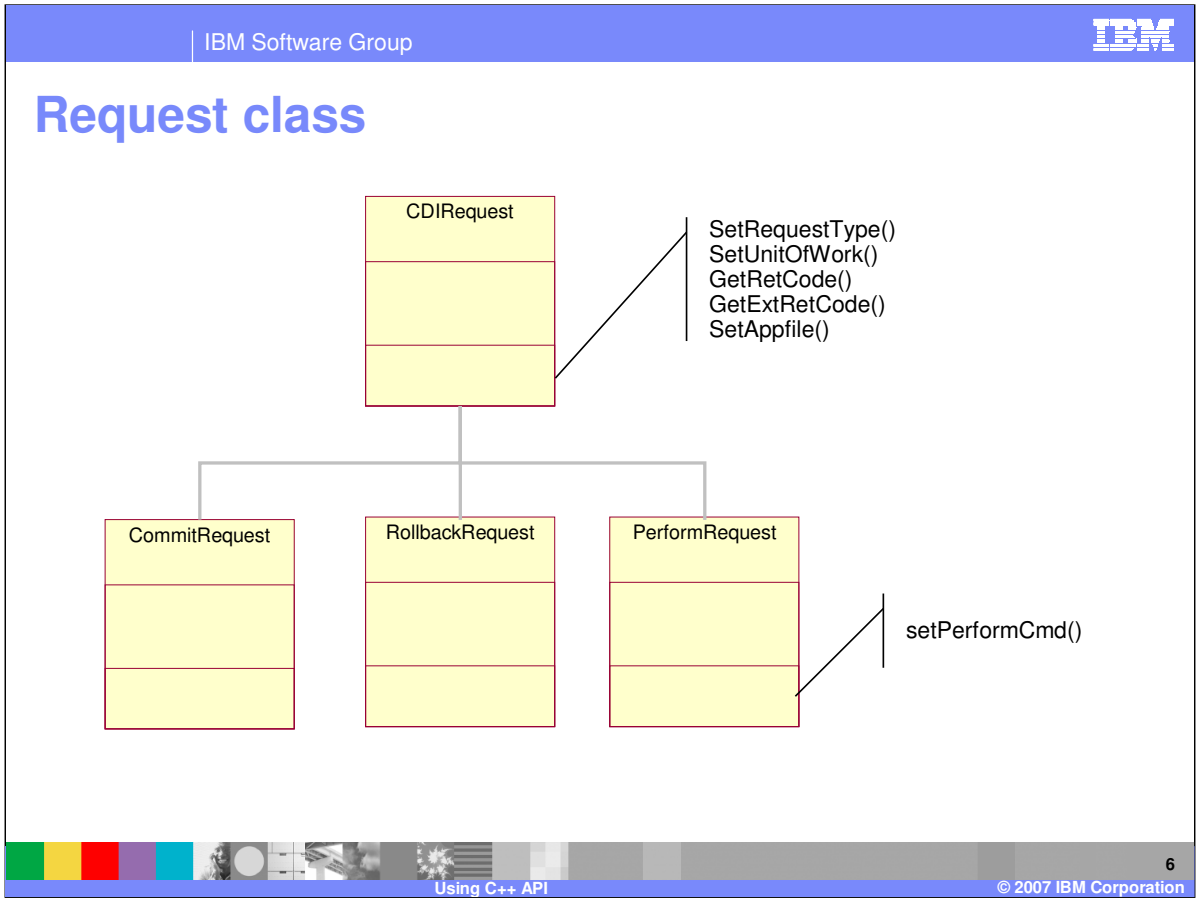
Environment class



An instance of the CDIEnvironment environment class must exist in order to initialize a translator. The methods of this class allow you to set DB2 connection information such as the subsystem identifier, plan name, user name and password. The Parse() method allows you to specify these configuration options in an array of option / value pairs consistent with the way parameters are passed from the operating system command interface.

For example, the following arguments might be passed to the Parse() method to facilitate connecting to a local DB2 system.

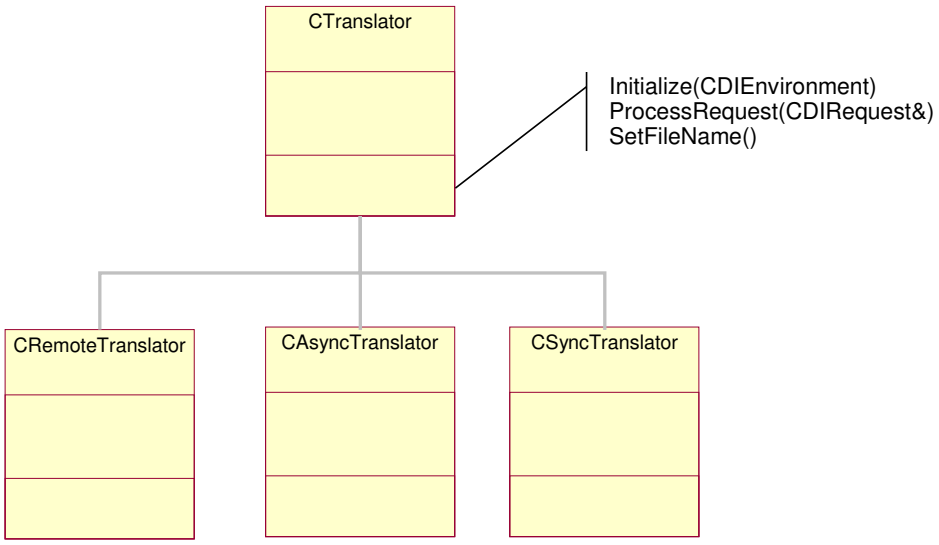
```
myApplication -user myuser -password myPassword
```



The CDIRequest class defines the action requested from the translator. This chart show some of the sub-types of this class: a commit, a rollback and a perform. The “perform” type contains a complete lexical statement compatible with the command interface specified in the Programmer’s Reference Guide. The “request” class offers “setter” function for many of the logical files used during translation such as the SetAppfile() method. These provide a simple way to supply logical file names without directly changing the PERFORM command, and they guarantee a syntactically correct request for the translator.

The “getter” methods give access to translation status. They also provide information about physical output files used during translation. The “request” may omit the specification for an output file, or it may specify a directory or relative path name. After translation, these attributes contain the full path names of files actually used. These same methods may return the number of characters that were read from or written to the logical file.

Translator class



All work is executed by a “translator” object. The application initializes the “translator” with a valid “environment” object, and it invokes operations through the “ProcessRequest()” method. The “SetFileName()” method builds any non-standard logical to physical file associations that can not be set in the “request” object.

WebSphere® Data Interchange utilities always use the CSyncTranslator for robustness and security.

The API in action: startup

```
#include “diapi.h”

   CDIEnvironment   aCDIEnvironment;
   CDIRequest       aTransformRequest;
   CSyncTranslator  aCSyncTranslator;

 //Define the Data Interchange Environment
   aCDIEnvironment.SetPlan("EDIEC33E");
   aCDIEnvironment.SetLang("ENU     ");

  rc = aCSyncTranslator.Initialize(aCDIEnvironment);
```


This begins a discussion of the sample program “apiexamp.cpp” code supplied with WebSphere® Data Interchange.

An application program should include “diapi.h” that includes the several headers defining the structures used by the low level API. The application next instantiates an “environment,” “request,” and “translator” object. It then customizes its environment object and uses it to initialize the translator. With this done, the environment object could be destroyed.

The API in action: building the request

```
// Name the input and output files:
    aCSyncTranslator.SetFileName("PRTFILE", "sample.prt");
    aCSyncTranslator.SetFileName("XMLFILE", "poxml5sr.dat");


   // Set the perform commands to be executed:
   // XML-TO-EDI TEST CASE: ******************************

   aTransformRequest.SetPerformCmd
   ("PERFORM TRANSFORM WHERE INFILE(XMLFILE) OUTFILE(OUTFILE) "
      "SYNTAX(X) CLEARFILE(Y) XMLEBCDIC(N) TRACELEVEL(A2)");
```


The sample next build the logical to physical association between a file system entry, sample.prt, and the logical name “PRTFILE.” This might better be done using the “SetPrintFile()” method of the object “aTransformRequest,” but it is possible to build any logical to physical association directly in the “translator” class provided it is not overridden by the value in this or another CDIRequest object. Setting the “XMLFILE” must be done using the CTranslator method since this name is not significant in WebSphere® Data Interchange.

The next call establishes the PERFORM command.

The API in action: execution

```
// Ask the translator to process the EDI to ADF Request:
   rc = aCSyncTranslator.ProcessRequest(aTransformRequest);

// Let the user know what happened:
   cout << "returns:  rc=" << rc
        << ", zccbrc=" << aCSyncTranslator.GetRetCode()
        << ", zccberc=" <<
          aCSyncTranslator.GetExtRetCode() << endl;

 // Confirm the input and output names and print return codes:
   rc=aCSyncTranslator.GetFileName(&pszPhysicalName, "XMLFILE",
                       &lFileLen);
   cout <<"Input file was    : " << pszPhysicalName << ", "
        << lFileLen << " bytes written" << endl;
```


Once the “request” object is complete, the CSyncTranslator instance uses it to actually execute the processing. Notice how the program calls the “GetFileName()” method of the translator.

The API in action: output

```
XML to EDI sample transformation using the C++ API

Initialize the translator SyncTranslator::Initialize()
Initialize() returns:  rc=0, zccbrc=0, zccberc=0

CSyncTranslator::ProcessRequest() returns:  rc=0, zccbrc=0,
zccberc=0

Input file: C:\...\samples\poxml5sr.dat, -1901 bytes written
Output file C:\...\samples\sample.out, 681 bytes written
```


This is a subset of the output from an executing sample program. The print of the XMLFILE shows a negative character count in addition to the fully qualified file name. The negative indicates that the characters were input rather than output.

Building an application

- Samples show Microsoft Visual C++ 6.0 project
  - links with ../bin/libdiapi.lib
- AIX samples with “apiexamp.mk” make file
  - links with ../bin/libdiapi.a

The sample directories on Windows and AIX contain files to build the sample application. On AIX, this is a basic ‘make’ file that sets the include path and brings in the library ‘libdiapi.a’ during link. These same functions are contained in a Microsoft Visual C++ project and workspace file for building the sample on Windows.

Summary

- WebSphere® Data Interchange has three C++ classes that give access to all of its features
  - CDIEnvironment
  - CDIRequest
  - CTranslator
- Features of the API are documented in the Programmer's Reference Guide
- The ../samples folder has a complete working example

The WebSphere® Data Interchange C++ API is simple and easy to use. It provides convenient access to the rich features of the product through three base classed: the environment, the request and the translator. These classes and their methods are documented in the Programmer's Reference Guide, and they are illustrated in "apiexamp.cpp" distributed in the samples folder on Windows and AIX.

Template Revision: 04/25/2006 11:09 AM

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	WMQ	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.