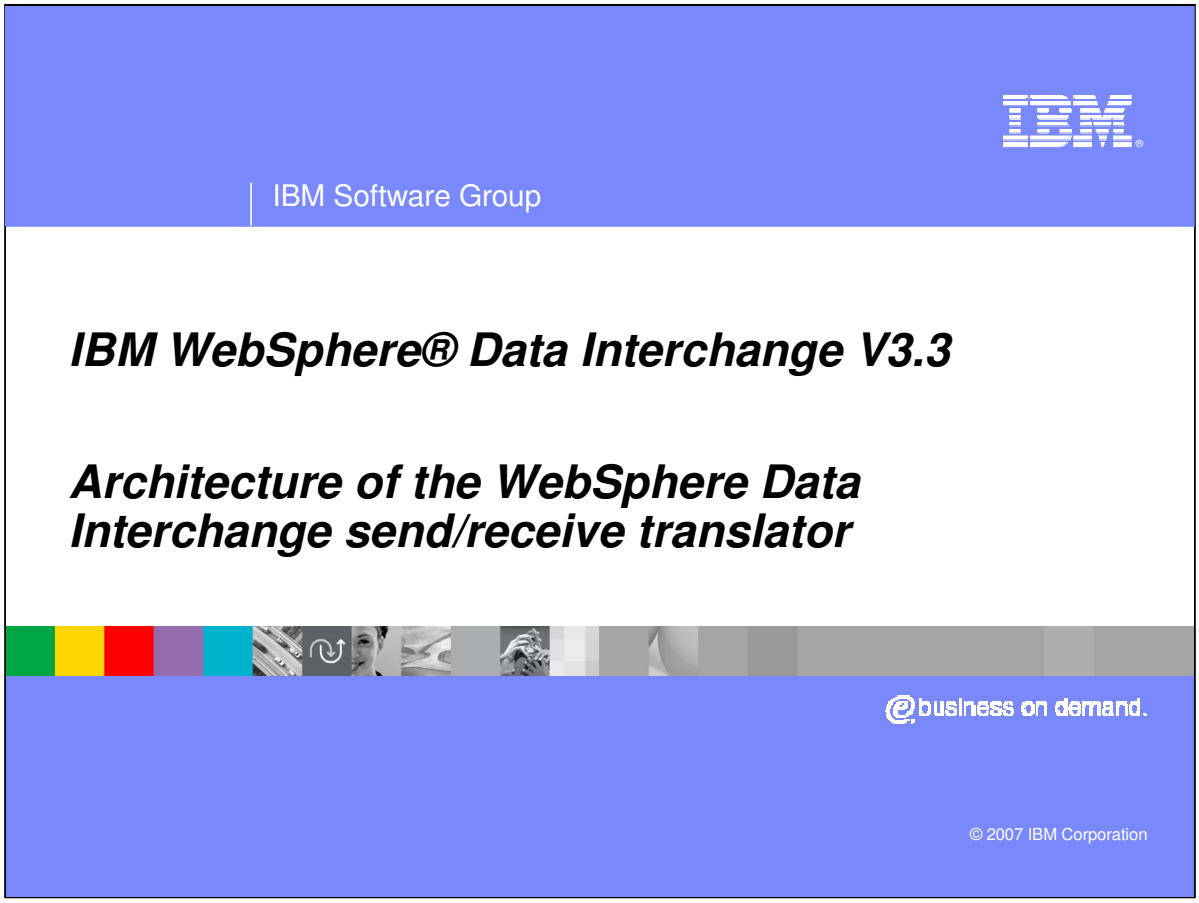


This presentation will describe the WebSphere Data Interchange Send/Receive Translator architecture.

Agenda

- Describe the WebSphere Data Interchange send/receive translator
  - Send translation process
  - Receive translation process
  - Fixed-to-Fixed translation process
  - XML translation process
  - Application programming interfaces

The presentation will describe the processing options available.

Section

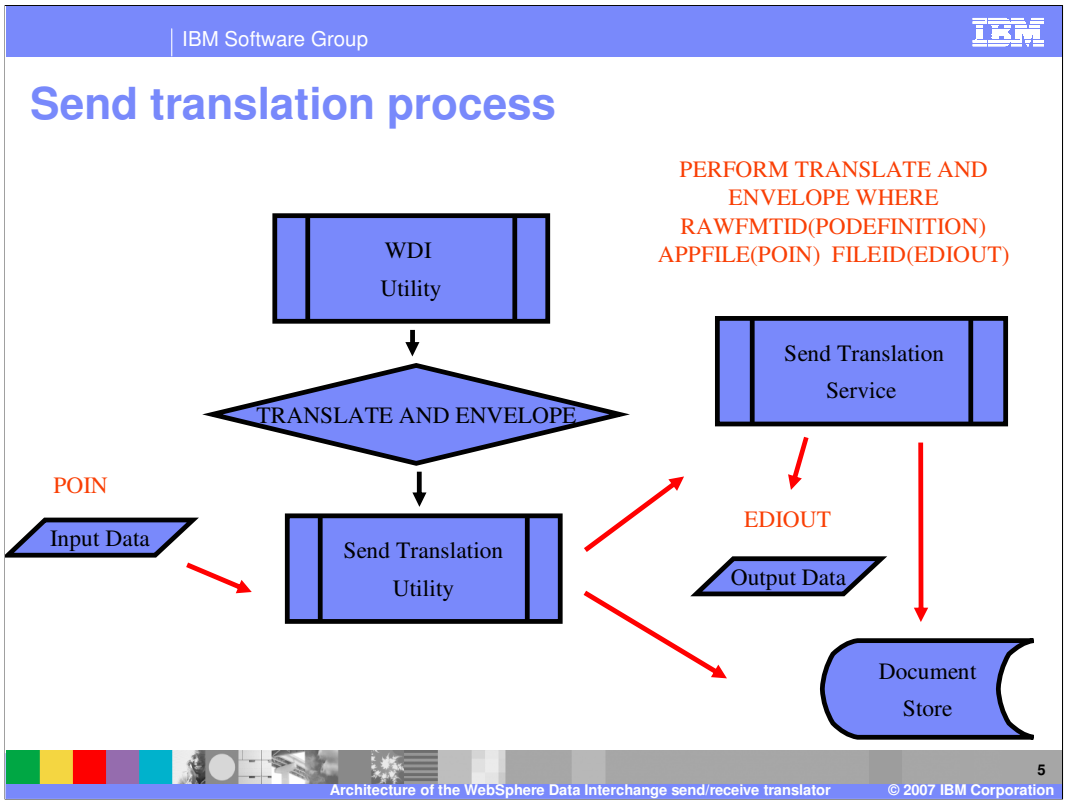
Send translation process

The Send Translation process.

Send translation process

- Source is application data
- Target is EDI
- Requires a WebSphere Data Interchange send map

The Send process must have application data as the source or input data type and the target or output data type will be EDI. A WebSphere Data Interchange Send map is required for this processing.



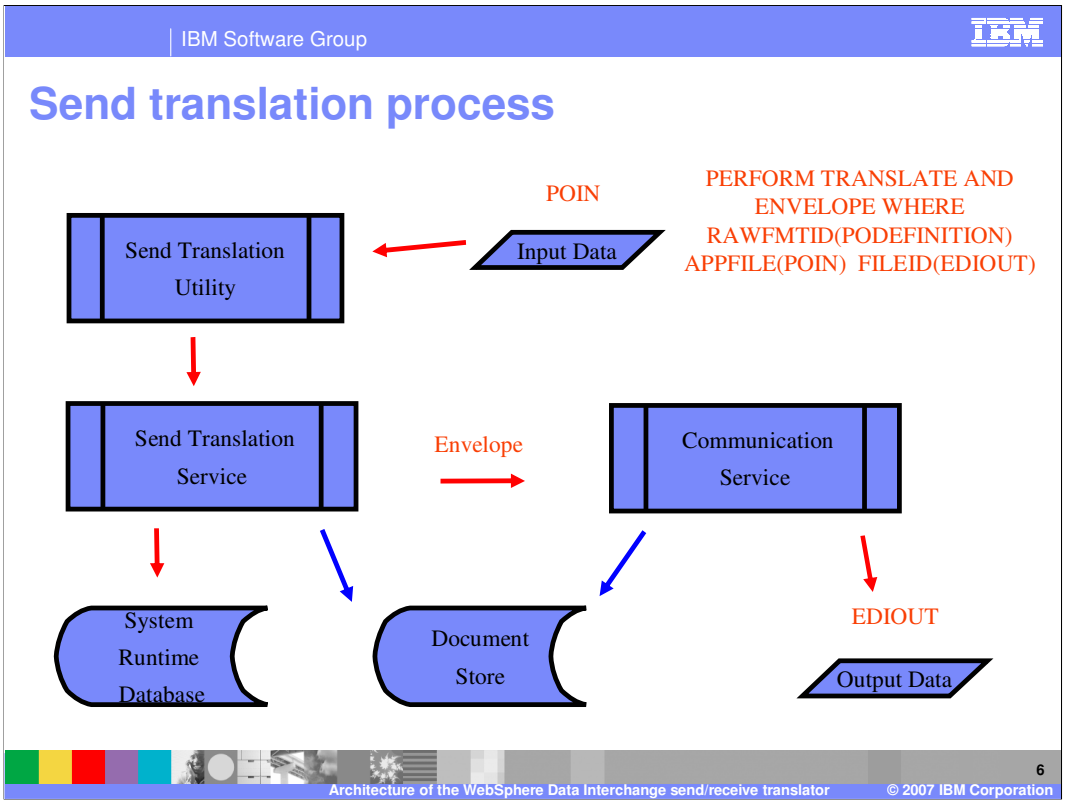
The send translation process includes the following functions:

Translating application data into an EDI standard format and placing it in the document store.

Enveloping standard transactions or messages so they are ready to be sent.

Sending enveloped data to trading partners.

The WebSphere Data Interchange utility provides command-level access to WebSphere Data Interchange Server services. These services can also be accessed using application programming interfaces (APIs) which perform the functions that the WebSphere Data Interchange utility provides.



There are two types of application data input RAWDATA and Control and Data or C&D format.

With RAWDATA, the RAWFMTID keyword is required when using the WebSphere Data Interchange Utility to identify the application data metadata definition. The metadata definition contains a beginning or ending record identification and internal or application trading partner identification. These values are extracted to find the send map usage which is attached to the send map to be executed during translation. Optional values such as the EDI and application trading partner may also be used to find a receive map usage.

With C&D format, the application data metadata definition is part of the data and is located on the C or Control record. The C record also contains the internal or application trading partner used to find the send map usage which is attached to the send map to be executed during translation. Each D or Data record contains the record name that is defined in the application data metadata definition.

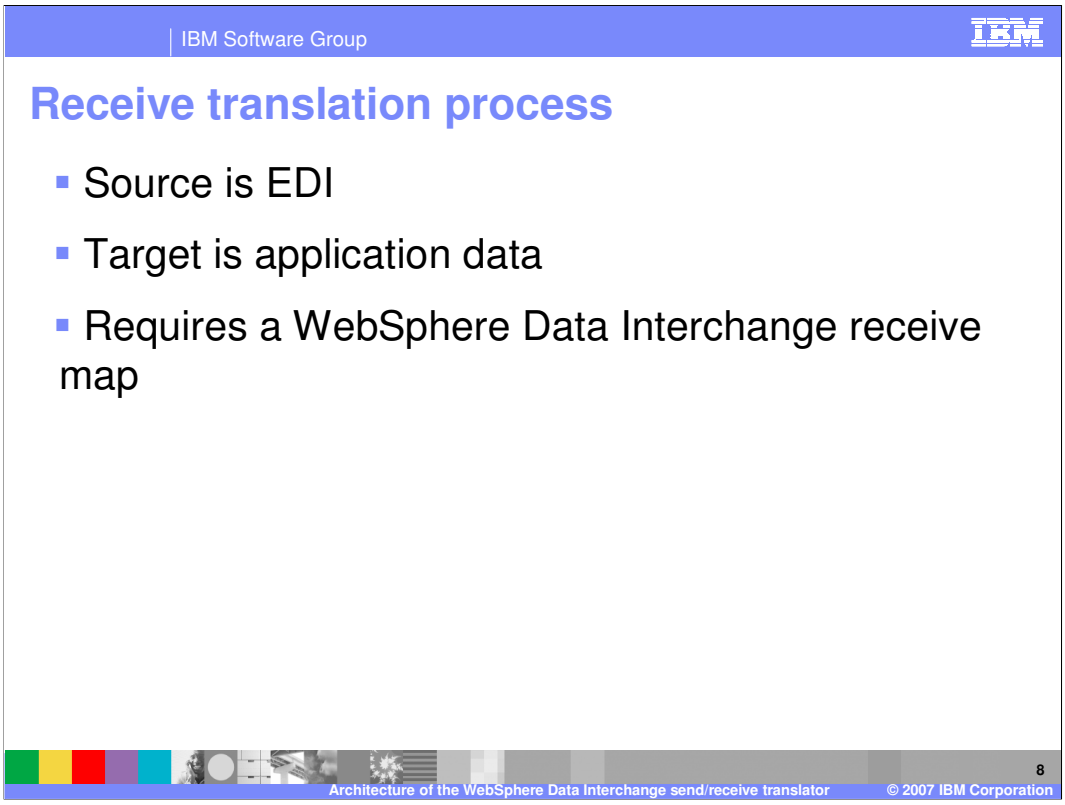
The Send Translation Utility reads the application input data and passes each record to the Send Translation Service.

The Send Translation Service reads the metadata definition from the system runtime database and identifies each record. Each record is stored until one logical message has been passed. The message is translated and

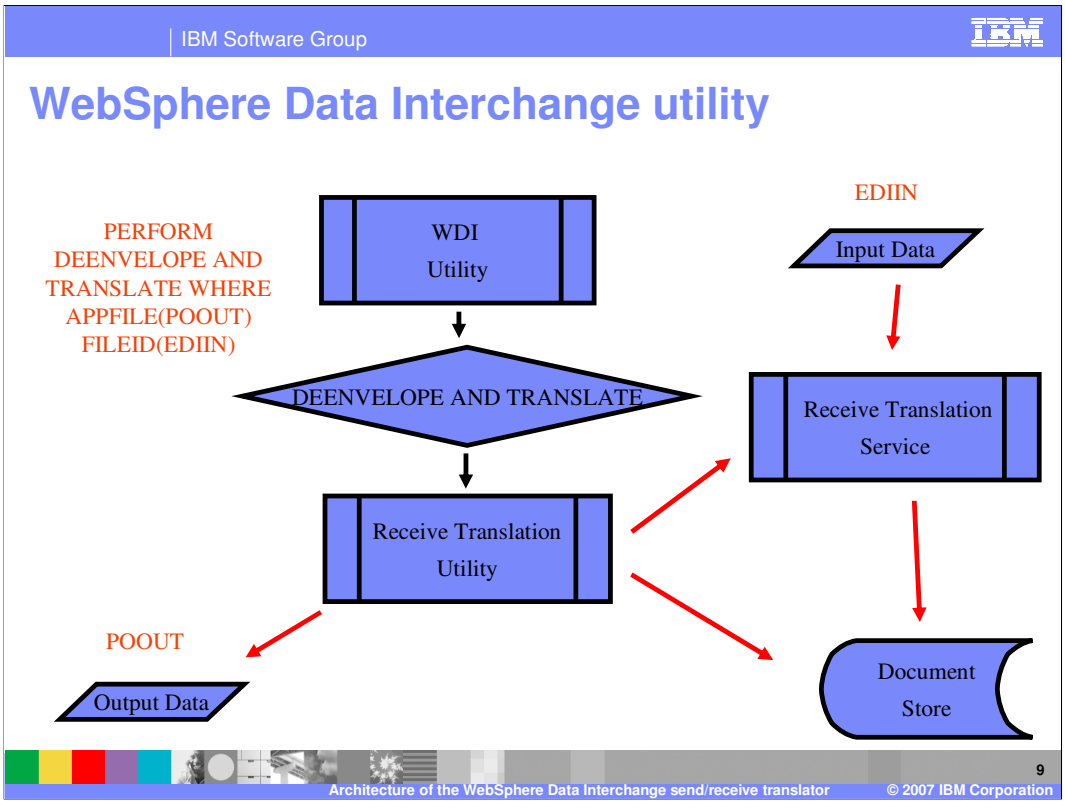
Section

Receive translation process

The Receive Translation process.



The Receive process must have EDI as the source or input data type and the target or output data type will be application data. A WebSphere Data Interchange Receive map is required for this processing.



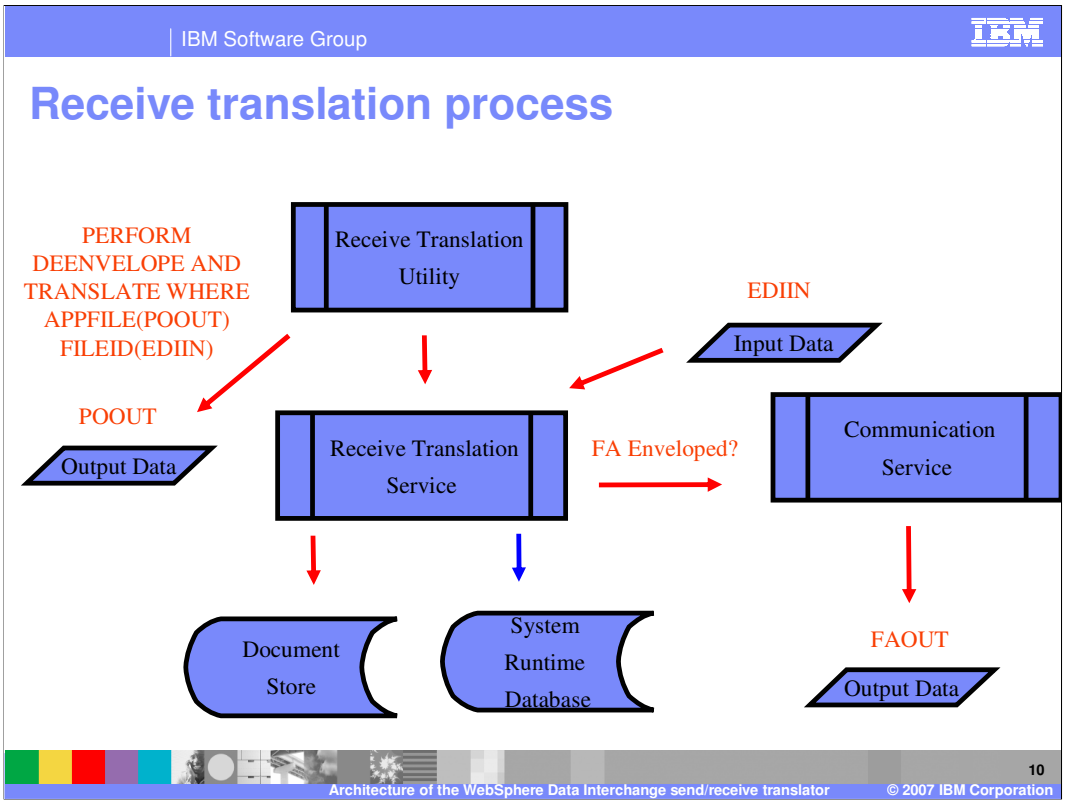
The Receive Translation Process includes the following functions:

Receiving data from trading partners.

Deenveloping interchanges and placing the EDI standard transactions or messages into the Document Store.

Translating EDI standard transactions or messages into application formats.

The WebSphere Data Interchange Utility provides command-level access to WebSphere Data Interchange Server services. These services can also be accessed using application programming interfaces (APIs) which perform the functions that the WDI Utility provides.



The Receive Translation Utility sets up control information for the Receive Translation Service.

The Receive Translation Service reads the input data and stores the data until one interchange has been retrieved. The interchange is deenveloped and the interchange sender ID and qualifier and the receiver ID and qualifier are extracted. These fields are used to find the sending and receiving trading partner profiles. The trading partner profiles along with the standard transaction or message type are used to find a receive map usage which is attached to the receive map to be used during translation execution. Optional values from the group envelope may also be used to find a receive map usage.

Each interchange is deenveloped and optionally translated depending on the utility PERFORM action. For example, PERFORM DEENVELOPE would not translate the messages inside the interchange whereas PERFORM DEENVELOPE AND TRANSLATE would translate. With each transaction or message within the interchange, the Receive Translation Service returns the application output data to the Receive Translation Utility. The Receive Translation Utility writes the application data to output.

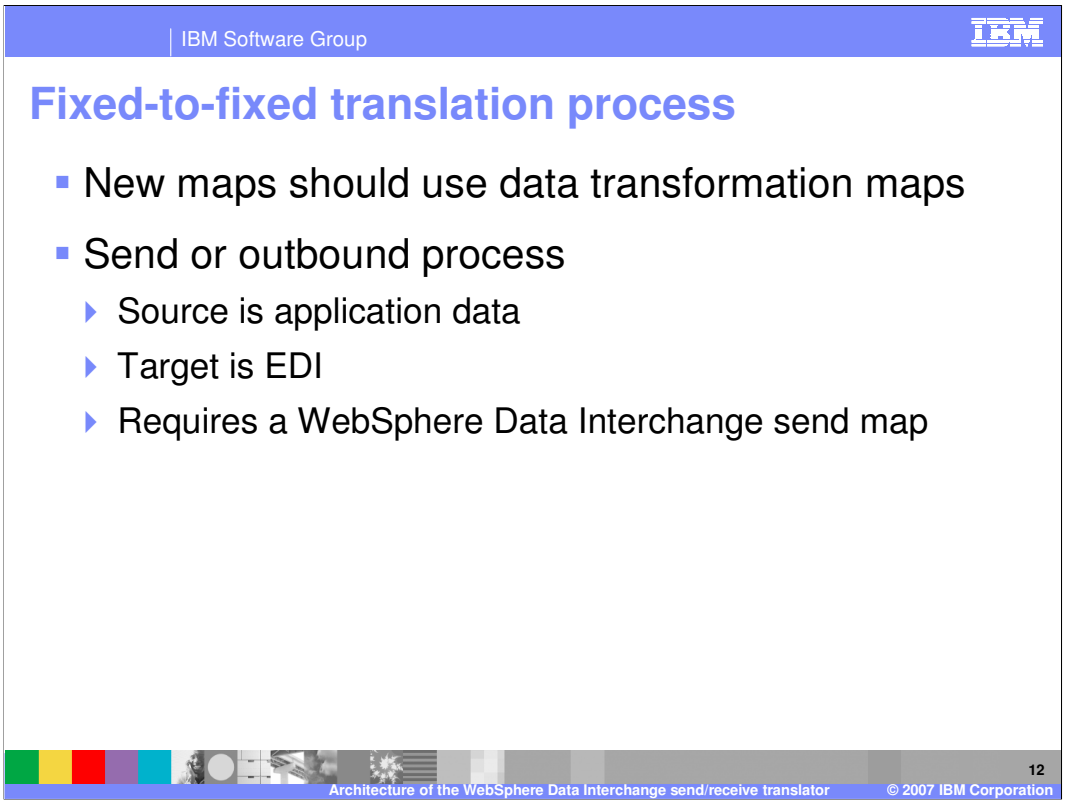
The Receive Translation Service and Receive Translation Utility work together processing each message until the input has been processed.

If Functional Acknowledgment (FA) processing and enveloping is to take

Section

Fixed-to-fixed translation

The WebSphere Data Interchange Fixed-to-Fixed Translation Process.



Fixed-to-fixed translation using WebSphere Data Interchange Send maps remains available for use with existing maps in WebSphere Data Interchange Version 3.2, but all new maps should be translated using the Data Transformation Process. The Data Transformation Process for this type of translation is a more user friendly process and does not require the setup and processing required when using the Send process described here.

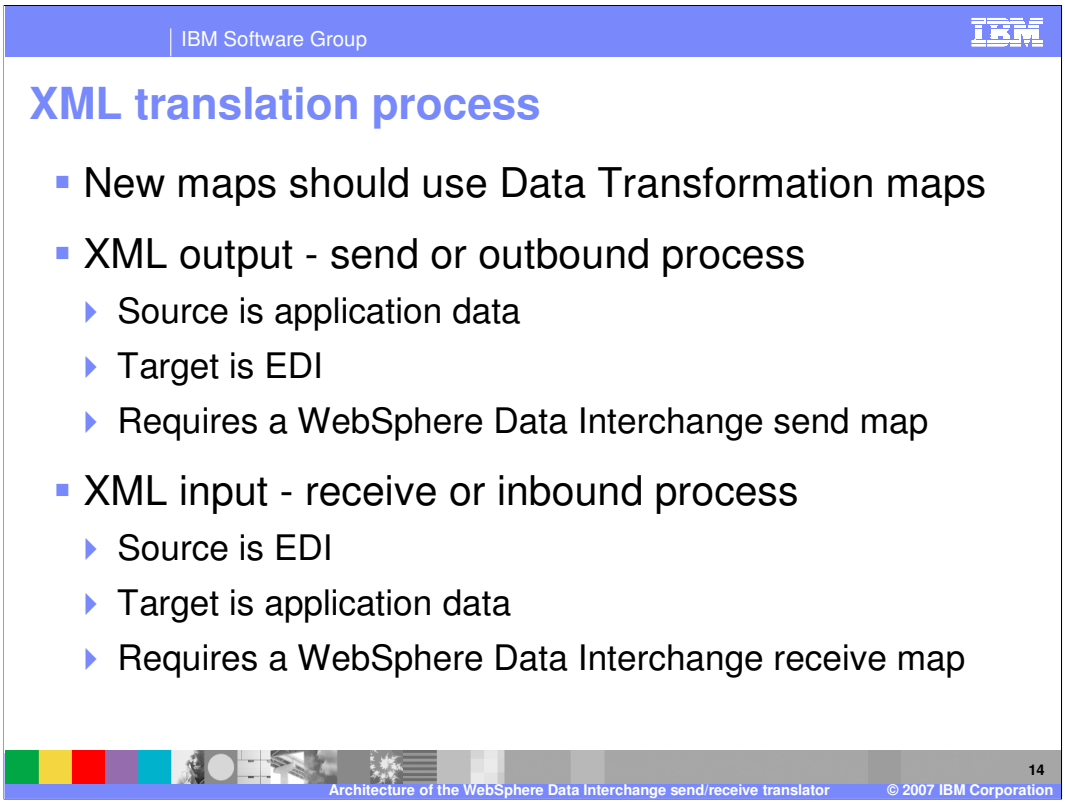
Fixed-to-Fixed Translation is an implementation for application data to application data translation. The same outbound processes used for Send translations are also used for fixed-to-fixed translation and enveloping. The Send map definition contains the source application data metadata. The target is an EDI representation of the target or output application data.

During the mapping process the source application data is defined and the target application data is defined. The WebSphere Data Interchange Client implements a function to create the EDI standard representation of an application data format definition. Performing this function on the target application data definition will create an EDI standard representation to be used with the mapping process. More information for Fixed-to-Fixed mapping can be found in the WebSphere Data Interchange V3.3 Mapping Guide.

Section

XML translation

The WebSphere Data Interchange XML Translation Process.



XML translation using Send and Receive maps remains available for use with existing maps in WebSphere Data Interchange Version 3.2, but all new maps should be translated using Data Transformation. The Data Transformation Process for this type of translation is a more user friendly process and does not require the setup and processing required when using the Send Receive process described here.

XML Translation is an implementation for XML data translation. The same outbound and inbound processes used for Send and Receive translations are also used for XML translation, enveloping, and deenveloping. The Send map definition contains the source application data metadata. The target is an EDI representation of the target or output XML data. The Receive map definition contains the source EDI representation of the XML data. The target is the application data metadata.

An XML DTD conversion utility is supplied to convert the XML DTD and create the EDI representation to be used during mapping. During the mapping process the source or target is XML as defined in the EDI representation. More information for XML mapping can be found in the WebSphere Data Interchange V3.3 Mapping Guide.

The WebSphere Data Interchange Utility provides keywords to identify that XML processing using a Send or Receive map is required. More information for XML processing and keywords can be found in the WebSphere Data Interchange V3.3 Utility Commands and File Formats Reference.

Section

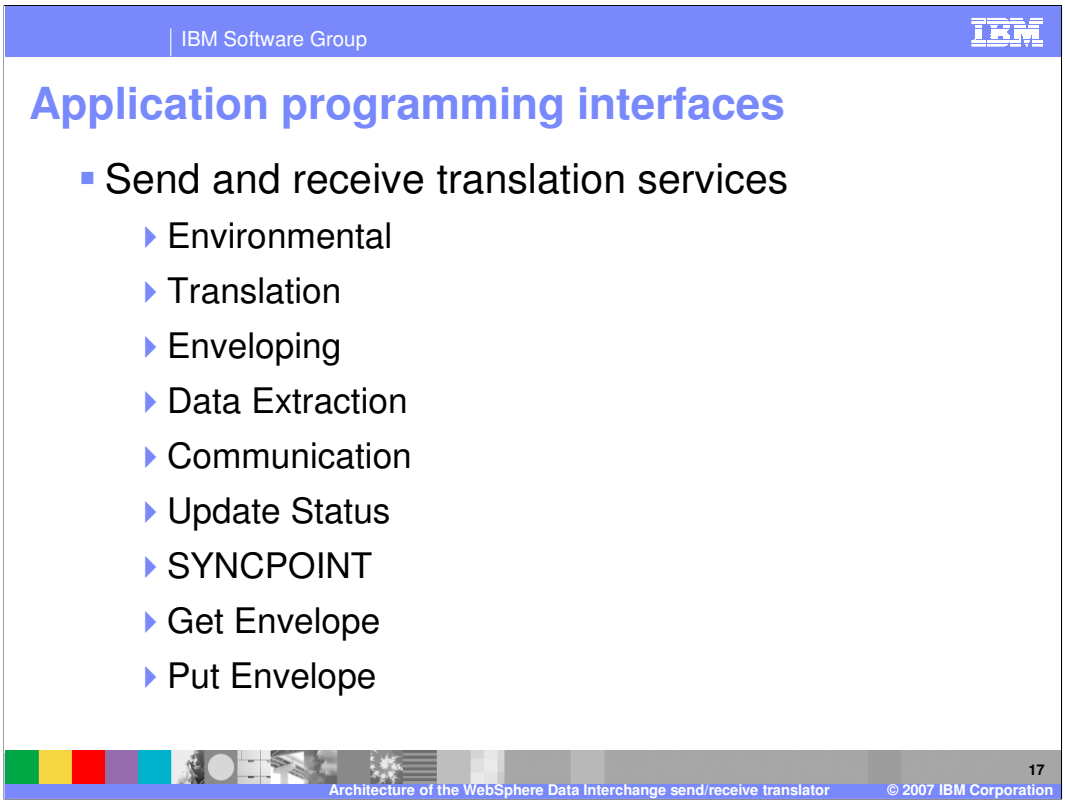
Application program interfaces

The WebSphere Data Interchange Application Program Interfaces.

Application programming interfaces

- Provides access to WebSphere Data Interchange Server services
  - Send and receive translation services
  - C++
  - JAVA
  - Utility

WebSphere Data Interchange provides application program interfaces (APIs) to allow your application direct access to the translation services. A number of Application Programming Interfaces are available for your processing needs. More information on using the interfaces can be found in the WebSphere Data Interchange V3.3 Programmer's Reference Guide.



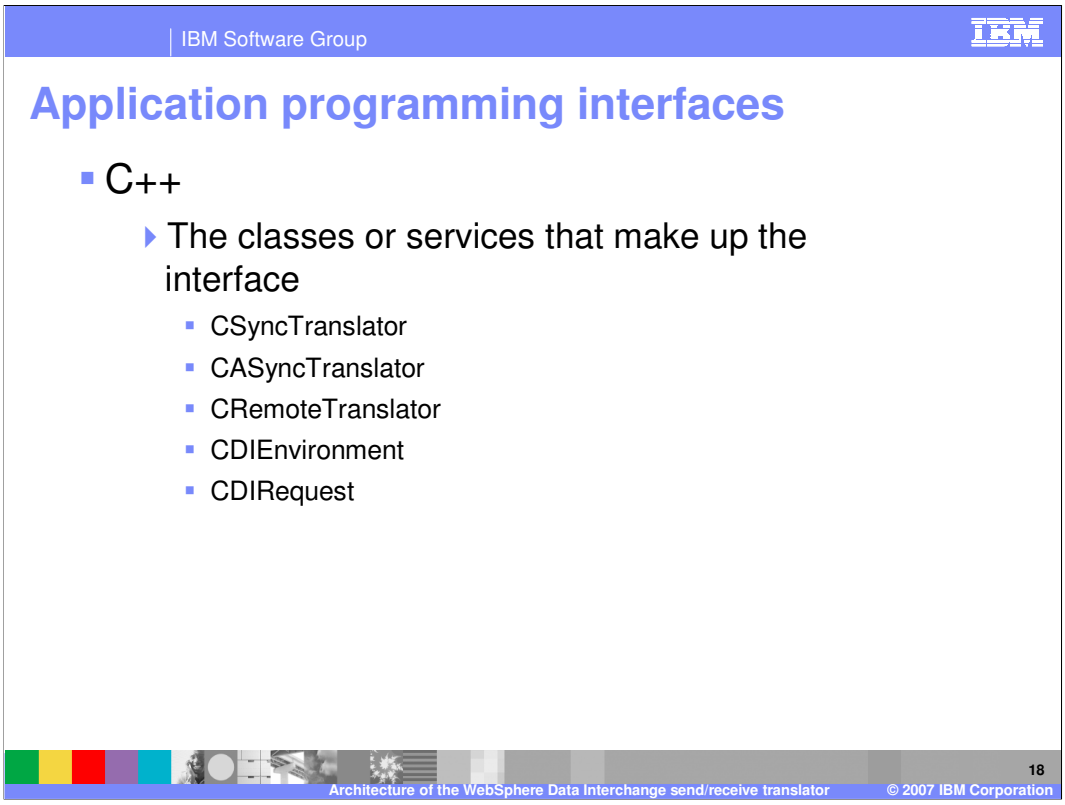
With the Send and Receive translation services, you can access WebSphere Data Interchange services with a simple call statement to a WebSphere Data Interchange-provided *stub* program. WebSphere Data Interchange provides four stub programs, one for each application programming language directly supported by WebSphere Data Interchange. A load library is distributed with WebSphere Data Interchange and contains a load module for each stub program. These stub programs are linked with the application program requesting the services. WebSphere Data Interchange is not physically part of the application load module. You must use the stub programs to access the WebSphere Data Interchange Application Programming Interfaces.

The following services are included:

Environmental services which is an initialization function to enable your program to use the WebSphere Data Interchange environment.

Translation services, enveloping services, and data extraction services are closely related. They share an API implemented by the same logical service (TRANPROC). These services also share the Document Store.

Enveloping services. Enveloping is necessary so networks can identify the trading partner who must receive the data and so the translators can identify and verify the type of data being received.



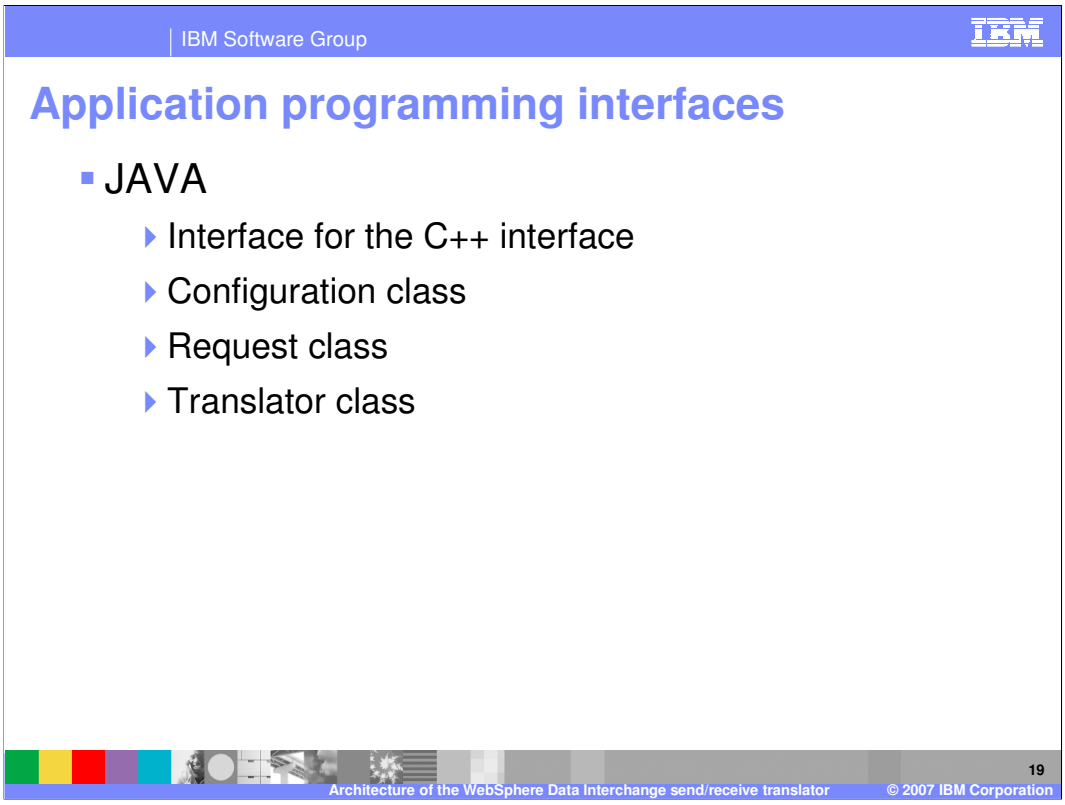
The C++ interface is made up of several classes that are all defined in the diapi.h header file shipped with the WebSphere Data Interchange product. When a program includes the diapi.h header file, it can use these objects to interact with the WebSphere Data Interchange translator by passing in PERFORM commands to either a CSyncTranslator, CASyncTranslator, or CRemoteTranslator object. The three different types of translator objects that can be created are as follows:

The CSyncTranslator provides access to the translator in a synchronous manner. The process waits for each command to complete before allowing the next command to be performed.

The CASyncTranslator provides asynchronous access to the translator. This method allows your program to begin processing on several transactions at once without waiting for the previous PERFORM command to complete.

The CRemoteTranslator provides access to a WebSphere Data Interchange translator running on a remote system. The CRemoteTranslator is like the CAsyncTranslator, except its constructor takes the hostname of the remote system as an additional argument to its constructor.

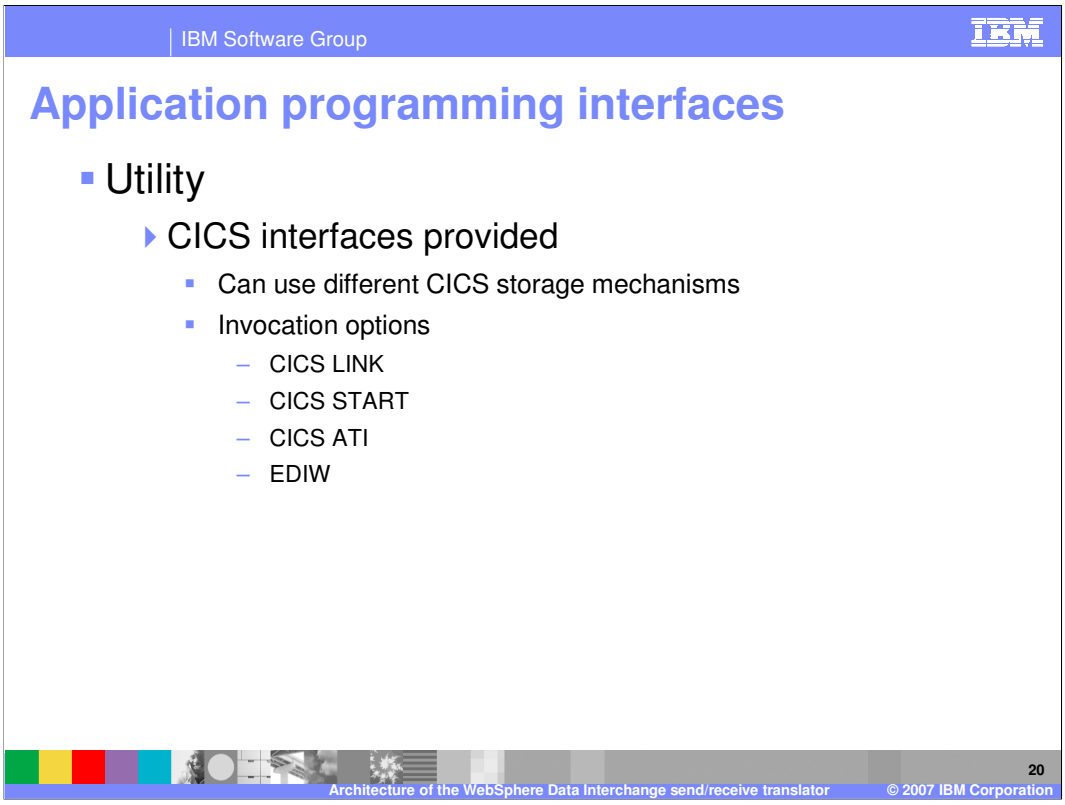
The CDIEnvironment class encapsulates all the system settings needed by the CSyncTranslator, CAsyncTranslator, and CRemoteTranslator during their initialization. The CDIEnvironment class must be instansiated and then passed to the initialize method of one of the translator objects.



WebSphere Data Interchange V3.3 has Java Native Interface implementation for its WebSphere Data Interchange C++ interface. A Configuration class is provided that encapsulates all the system settings needed by the Translator class during their initialization. The Configuration class must be instantiated and then passed to the initialize method of Translator objects.

The Request class is the request for translation that will be submitted to a translator. The Request object names the files necessary to perform a translation as well as the perform statement to be executed.

The Translator class provides access to the translator in a synchronous manner. The process waits for each command to complete before enabling the next command to be performed.



In CICS, when you develop an application program to run the WebSphere Data Interchange Utility, you can write the application in any programming language supported by CICS. The Utility provides standard CICS interfaces and can be instructed to use different CICS storage mechanisms.

The following methods can be used to invoke the WebSphere Data Interchange Utility:

Your application can use the CICS LINK command to link to the EDIFFUT program. This is the simplest interface you can use. With this command, the WebSphere Data Interchange Utility executes synchronously with your application. When the WebSphere Data Interchange Utility finishes running, it issues a CICS RETURN command to give control back to your application.

Your application can use the CICS START command to start the CICS transaction EDIB. The WebSphere Data Interchange Utility executes in a separate CICS transaction, asynchronously with your application. If you use the CICS START command, consider using response applications, because they are an essential element of the entire process.

To use the CICS Automatic Task Initiation (ATI) command, you must first add an intrapartition TD queue to the Destination Control Table (DCT) and associate a trigger level and transaction EDIB with the TD queue. As with the CICS START interface, the WebSphere DataInterchange Utility executes in a separate CICS transaction, asynchronously with your application.

References

- WebSphere Data Interchange V3.3 Programmer's Reference Guide

More information can be found in the WebSphere Data Interchange Version 3.3 Programmer's Reference Guide.

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	WebSphere MQ	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.