SYBASE®

Developer's Guide

e-Adapter Development Kit

Version 3.9

DOCUMENT ID: DC33126-01-0390-02

LAST REVISED: November 2004

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

The e-Adapter Development Kit (e-ADK®) provides components for creating interface adapters with minimum duplication and maintenance. The e-ADK is a set of tools and libraries designed to simplify the development of adapters that interface with web-based, packaged, and legacy applications. This provides a standard approach and framework for developing custom adapters. The e-ADK provides consistent functionality and a common user interface across platforms.

Audience

Adapter developers are targeted as the primary users of this book. The adapter developer who uses this book to develop an adapter must also be familiar with the contents of the *e-ADK Programmer's Reference* and the *e-ADK Installation Guide*.

How to use this book

The *e-ADK Developer's Guide* provides an overview of the e-ADK. This guide also defines the e-ADK files and functions that the developer can modify.

The guide is organized into the following chapters:

- "About This Book" provides a brief introduction to e-ADK 3.9, a list of available documentation, and technical support information.
- Chapter 1, "Architectural Overview," provides an overview of the e-ADK adapter architecture, tools, transport layer, and the development process.
- Chapter 2, "Modes," provides an in-depth presentation of the modes used for adapters.
- Chapter 3, "Adapter Development Process," provides the steps for developing an adapter using e-ADK 3.9.
- Chapter 4, "Adapter Runtime Environment," provides information about configuring the environment; registering an NT service; testing, deploying, and executing the adapter; and exception handling.
- Chapter 5, "Troubleshooting," provides additional information to help with the development of adapters.

Related documents

This section describes the documentation available for the e-Adapter Development Kit, release 3.9.

Cross-Platform Documentation The e-ADK documentation set for developers consists of the following documents:

- *Installation Guide*
- *Developer's Guide*
- *Programmer's Reference*
- *Feature Guide*

Related Documentation The following Open Transport Version 2.6 documents are referenced in this document set to supply you with specific information that supports this product:

- *EMQ Driver Configuration Guide*
- *File Driver Configuration Guide*
- *MQ Series Driver Configuration Guide*
- *MSMQ Driver Configuration Guide*

Other related documentation is available from New Era of Networks, Sybase, and IBM. Refer to other documentation from each of these companies for more detail about use of applications relevant to this product.

To access the Technical Library Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals.

Other sources of information

Use the Sybase Getting Started CD, the SyBooks Bookshelf CD, and the Sybase Product Manuals web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks Bookshelf CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks Bookshelf CD is included with your software. It contains product manuals in a platform-independent bookshelf that contains fully searchable, HTML-based documentation.

  Some documentation is provided in PDF format, which you can access through the PDF directory on the SyBooks Bookshelf CD. To view the PDF files, you need Adobe Acrobat Reader.

Refer to the *README.txt* file on the SyBooks Bookshelf CD for instructions on installing and starting SyBooks.

- The Sybase Product Manuals Web site is the online version of the SyBooks Bookshelf CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

1. Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.
2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.
4. Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the "Technical Support Contact" role to your MySybase profile.
5. Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Conventions

The following formatting conventions are used in this manual:

Formatting example	To indicate
command names and method names	When used in descriptive text, this font indicates keywords such as: • Command names used in descriptive text • C++ and Java method or class names used in descriptive text • Java package names used in descriptive text

Formatting example	To indicate
	Italic font indicates:
myCounter variable	• Program variables.
Server`.log`	• Parts of input text that must be substituted.
myfile.txt	• File names.
User Guide	• Book titles.
sybase/bin	Directory names appearing in text display in lowercase unless the system is case sensitive. A forward slash ("/") indicates generic directory information. A backslash ("\") applies to Windows users only.
"About This Book"	Chapter titles have initial caps and are enclosed within quotation marks.
File > Save	Menu names and menu items are displayed in plain text. The angle bracket indicates how to navigate menu selections, such as from the File menu to the Save option.
`parse|put|get`	The vertical bar indicates you may select only one of the options shown in the code.
	Monospace font indicates:
`create table`	• Information that you enter on a command line or as program text.
`table created`	• Example output fragments.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

Architectural Overview

This chapter provides an overview of e-ADK and the architectural structure of the product.

Purpose

New Era of Networks adapters are a non-invasive way to integrate systems. An adapter serves as the entry and exit points that connect a source or target application data with any one of the integration servers using New Era of Networks technologies: New Era of Networks Rules, New Era of Networks Formatter, e-Biz Integrator, IBM MQ Series Integrator, and IBM WebSphere Integrator. New Era of Networks adapters also work with the following application servers: Sybase EAServer, IBM WebSphere Application Server, and BEA WebLogic Application Server.

The e-ADK, which provides you a way to minimize development and maintenance efforts, is a specialized C++ software development kit (SDK) used to build adapters. The e-ADK contains the SDK and the adapter runtime environment (ARE). You send only the ARE component with your completed adapter.

Additionally, e-ADK, release 3.9, uses open transport, which provides a transport and transaction manager independent interface for the application layer. Because the interface is the same regardless of the underlying transport, applications do not require a rewrite or recompile to accommodate a new transport.

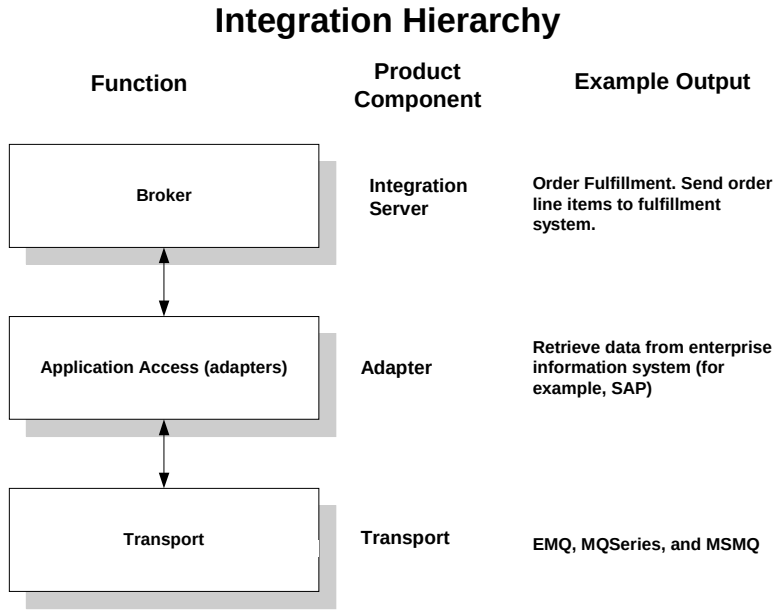
The e-ADK provides the following benefits:

- Consistent functionality.
- A common user interface across platforms.
- Simplified development of adapters that interface with web-based, packaged, and legacy applications.
- Accommodation of a new transport without a rewrite or recompile.
- Developer-customized adapter-specific error handling and exception handling.

Integration

Integrating applications requires many pieces working together to create a synergistic whole. New Era of Networks provides much of the infrastructure necessary to achieve that whole of which adapters are a piece. The following sections give you an overall perspective of the various pieces and their division of responsibilities. Additionally, they describe the skills and knowledge base that are required for various levels of integration and automation. The following graphic illustrates the integration of the e-ADK parts.

Figure 1-1: e-ADK Integration



Integration Server

To integrate systems that were never intended to work together in a scalable and manageable way, the formats (or the language of the specific application) must be understood by the other systems or another source must translate that format from the source application to the destination application. This is the function of the integration server. The integration server provides a loosely coupled interaction between applications. The message parsing, enrichment, transformation, and routing capabilities are encapsulated within the integration server. Note that the integration server does not keep any context for messages that are processed. Thus, it has high throughput capabilities. This is typically called a fire-and-forget model.

Adapter

An adapter provides entry and exit points for an integration server. Adapters are responsible for knowing the application metadata, its data models, extracting and delivering data from the application at run-time, exception handling, and handling any error conditions while interacting with the application. Modifying the adapters by modifying the code is more burdensome and costly than declaring the requirements in the integration server.

Adapters perform the following functions:

- Obtain metadata from external data sources to a message broker format repository or document type definition (DTD).
- Read data from sources such as applications, files, databases, and data streams, and place the data on a message transport.
- Write data to targets such as applications, files, databases, and data streams, after getting the data from a message transport. See the following section for additional information about message transport.

Transport

Transports convey messages between the adapter and a source or target. The following are examples of transports used with New Era of Networks adapters:

- IBM MQ Series
- IBM WMQI
- Microsoft MSMQ
- New Era of Networks Enhanced Message Queueing (EMQ)

A reliable means of transmitting information has become a necessity for conducting electronic business. This is accomplished by message-oriented middleware such as those listed above. Transports provide guaranteed delivery, platform portability, asynchronous messaging, and transactionality. To ensure that messages are not lost when errors occur, e-ADK 3.9 allows transactions to be rolled back when errors in transport are encountered.

To accommodate internationalization, when you are using a multi-byte character encoding set, use RFH2 headers.

Rather than requiring in-depth knowledge of the transport method used to transport messages, e-ADK 3.9 provides classes that allow you to identify transport through a simple configuration key.

Terminology

The following terms are applicable to adapters:

Application A packaged application, database, protocol, file, or other data source.

Snap-In A software component that provides easy access and configuration of information from the framework of the Microsoft Management Console for Windows NT. You can develop a snap-in as a stand-alone, extension, or dual-snap-in.

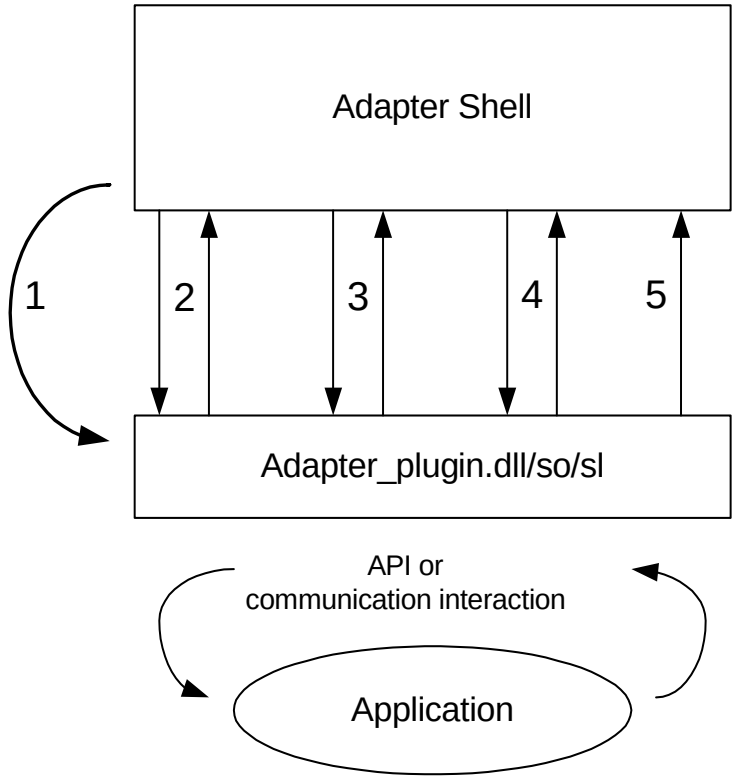
Plug-In An external software or SQL program that is accessed by a larger application to provide added and customer-specific functionality.

Mode The means e-ADK and adapters built on e-ADK use to invoke a specific method of operation.

e-ADK Architecture

The e-ADK has two components: the Adapter Shell and an adapter plug-in. The Adapter Shell provided by the e-ADK dynamically loads and makes calls to an adapter plug-in. The adapter plug-in, which interacts with a target and an application or data source, is built by the developer using the APIs provided with the e-ADK. Detailed API infomraiton is available in the *e-ADK Programmer's Reference.* The plug-in contains the adapter library. The following graphic shows high-level interaction between the Adapter Shell and the adapter plug-in:

Figure 1-2: e-ADK Architecture



Adapter Shell

The e-ADK provides an executable that contains libraries and classes that run the adapters. The executable functionality is sometimes called Adapter Shell. The Adapter Shell provides a consistent processing flow for all adapters, for example, similar command line, configuration, and processing. The Adapter Shell performs general initialization and setup before calling a function and then cleans up before shutdown. The following numbers correspond with the numbers on the graphic in the previous section.

1 The Adapter Shell loads the adapter plug-in.

2 The Adapter Shell initializes itself.

- Determine the mode and data representation. The modes are Acquire, Deliver, Process, Schema, Schema Remove and Catalog. Data representation is either tree (NDO) or buffer.
- Initialize input and/or output transports and open the transports for Acquire, Deliver, and Process modes.

3 The e-ADK calls initAdapter() in the adatper plug-in. You must provide the configuration information as a parameter in the adapter plug-in. Based on the invocation, the mode is executed in a loop with the appropriate plug-in function called and the transport or schema functionality performed.

4 The e-ADK calls functions in the adapter plug-in.

5 The e-ADK calls shutdownAdapter() in the adapter plug-in. You must provide the configuration information as a parameter in the adapter plug-in. Processing finishes and shutdown is called.

6 Adapter Shell throws and handles exceptions as they occur.

Adapter Plug-In

The adapter plug-in performs the interaction with an application and data sources/targets. The Adapter Shell loads the adapter plug-in at initialization. The adapter plug-in is responsible for processing and providing the necessary data.

Modes

Modes are used to direct the actions of the Adapter Shell. The e-ADK supports Schema, Catalog, Schema Remove, Acquire, Deliver, and Process modes.

- Schema is used to create formats for storing data structure definitions in a repository. Run the acquireSchema function to define all formats that are used to parse and reformat messages. Schemas can be created in NCF, DTD, or XML formats.
- Catalog is used to get listings of supported schemas from the adapter.
- Schema Remove is used to remove formats from a repository. Mapping information is retained in the repository when the formats are removed.

- Acquire is used to get data from an application and put that data to a transport.
- Deliver is used to get information from a transport and deliver it to the application.
- Process is used to get data from a transport, call the user-written processing function, and then deliver data to an output transport. This effectively provides a shell for processing data.

For additional information about modes, see Chapter 2, “Modes.”

Data Representation

Adapters use two methods of transferring data: buffer and tree. The buffer method moves raw data to and from the transport with no additional processing. The tree method moves the data in an hierarchical (tree) structure. The term New Era of Networks Data Object (NDO) is the New Era of Networks-specific name given to the tree method of exchanging data.

Acquire, Deliver, and Process modes each handle both types of transferred data: buffer and tree. Schema handles only tree data. When using the tree methods, the Adapter Shell serializes and deserializes data to and from the transport. The basic difference between using a buffer or a data tree is that if your adapter plug-in returns or requires a data tree, the e-ADK will serialize or deserialize the tree for transportation purposes.

You must create the following functions, depending on whether you are using the buffer or tree representation. The Adapter Shell calls this function during processing.

Table 1-1:

	For Tree (NDO)	For Buffer
Schema	acquireSchema()	none
Catalog	acquireCatalog()	none
Schema Remove	none	none
Acquire	acquireData()	acquireBuffer()
Deliver	deliverData()	deliverBuffer()
Process	processData()	processBuffer()

NNADKStubPlugIn.cpp contains templates for buffer and tree functions. For additional inforamtin about using buffer and tree represnetaiton, see the *Programmer's Reference*.

CHAPTER 2

Modes

Mode is the invocation of a specific method of operation. The e-ADK supports six modes: Schema, Catalog, Schema Remove, Acquire, Deliver, and Process. Each launch of NNSYAdapter39.exe can handle only one mode at a time so a separate instance of NNSYAdapter39.exe is required to handle running more than one mode at a time. The mode in which NNSYAdapter39.exe is run is dependent on the configuration, which is explained in this chapter.

When building an adapter, you must address modes and data representation both during design and during runtime. See "Defining Your Adapter Plug-In" on page 40 for working with modes and data representation during design time and "Creating Configuration Files for Your Adapter Plug-In" on page 44 for runtime information. "Methods of Handling Errors" on page 77 provides information to help you understand how modes work with specific errors.

Sample configuration files are provided in the examples directory. You can use those files as a temple.

This chapter provides detailed information about the following modes:

Schema

Schema mode is used during design time to create formats for storing data structure definitions in a repository. Users must supply an acquireSchema() function in their plug-in library to define all schemas for the integration servers to use to parse and reformat messages. Each schema is equal to one NDO.

Warning! Follow the rules that govern the creation of XML names. See Chapter 8: "Understanding NDO Node Names" of the *e-ADK Programmer's Reference* for information about naming NDO nodes and colon restrictions.

Unlike Acquire, Deliver, and Process, the Schema mode works only with trees, not buffers. Schema builds a tree and stores the specified schema structure in a repository or an XML or DTD file depending on the values set for the Adapter.SchemaLoader.Factory and Adapter.SchemaLoader.Library keys. The acquireSchema() function is called after initialization takes place inside the adapter plug-in. The return value from this function defines whether the loop for acquireSchema() should continue or stop. The function is called in a loop until FALSE is returned so that multiple schemas can be defined in one run of the executable.

Note Consider supplying a Schema function only if you are using NDO data representation during run time.

Schema mode builds a schema tree and stores it in a repository. Data is retrieved in one of two ways. To get the schema:

- Retrieve the schema from a plug-in. In this case, the user writes the code that gets the schema.
- Write an acquireSchema function in the plug-in library.

  This function receives an empty NDO and a schema name from the e–ADK. The function should populate the NDO with the adapter's schema based on the schema name.

Schema Mode Internationalization

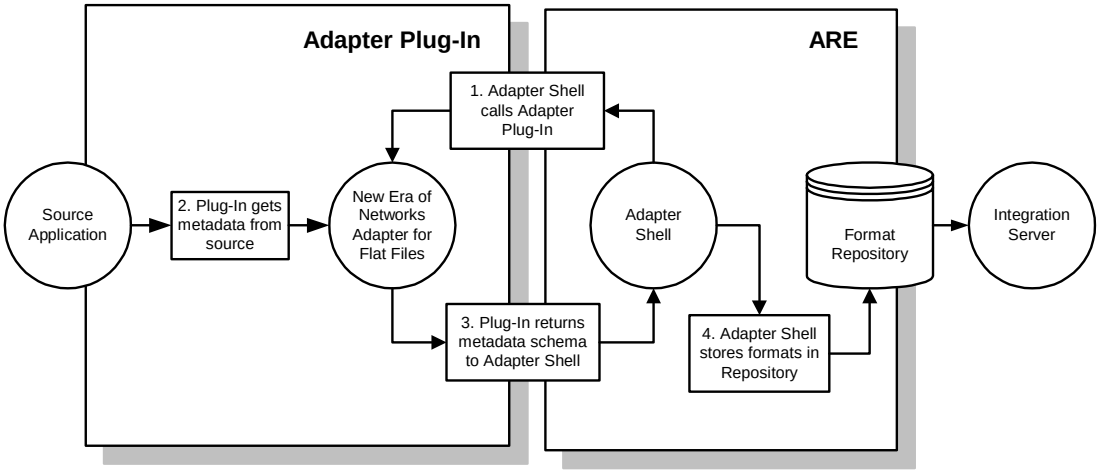
NDO Schema trees can be created in any system encoding. For NCF Schema loading, it is advisable to create NDO schema trees in the native system encoding. The database drivers and the database server software perform a character set negotiation when the schemas are loaded to the database. The database drivers use the native system encoding to establish their end of the negotiation. Attempting to load data of another character encoding set can result in improper translation. For NCF schema loading, the e-ADK examines the NDO schema tree and, if not in the native system encoding, converts it. Characters that are not convertible do not result in an exception being thrown. Instead, the non-convertible character is replaced with a fill character which will result in improperly loaded schema information.

NCF format prefixes must be ASCII-compatible single-byte characters. NCF format names, however, can be crated using multi-byte characters. The formatter database enforces a 120-byte maximum length for format names. If your format name is longer than 120 bytes and contains multi-byte characters, you must set the configuration key I18N_Condense equal to true. The condense routine truncates characters from the string. Setting I18N_Condense equal to false uses the old condensing algorithm which only works with single-byte data.

Schema Mode Diagram

The following diagram illustrates the Schema mode.

Figure 2-1: e-ADK Schema Mode



Schema Mode Example Configuration File

The following is an example configuration file with values set for Schema mode using NNTSchemaLoader:

```
### File Name:  schema.dat

Adapter
    test.drive=false
    adapter=nnadkstub
    mode=SCHEMA
    prefix=aaa
    repository.dir=c:\nnsy\NNSYContentRepository
    SchemaLoader.Factory=NNTSchemaLoader_Factory
    #for all database types
    session=ADKSession
    SchemaLoader.Library=adk39nnt56sl
    session=ADKSession
    clash.avoid=true
    continue.format.exists=true
    I18N_Condense=false
```

```
Session.ADKSession
     #for MSSQL
     NNOT_SHARED_LIBRARY     =dbt26sql65
     NNOT_FACTORY_FUNCTION   =NNSesMS6Factory
     #for Oracle 8
     #NNOT_SHARED_LIBRARY     =dbt26or806
     #NNOT_FACTORY_FUNCTION   =NNSesOra8Factory
     #for DB2
     #NNOT_SHARED_LIBRARY     =dbt26db250
     #NNOT_FACTORY_FUNCTION   =NNSesDB2Factory
     #for Sybase
     #NNOT_SHARED_LIBRARY     =dbt26syb11
     #NNOT_FACTORY_FUNCTION   =NNSesSybCTFactory

     #as required by the database type
     NN_SES_SERVER           =test
     NN_SES_USER_ID          =testuser
     NN_SES_PASSWORD         =userpswd
     NN_SES_DB_NAME          =dbms56
```

Catalog

Catalog mode is used to retrieve a list of all formats available for a specific adapter. This functionality is primarily used by New Era of Networks Adapter GUIs that provide configuration services and functionality for various New Era of Networks adapters. Catalog mode, however, can be used without the GUI to generate a listing of the available schemas that are written as a file to storage in XML format. This file can then be viewed or consumed by another application. You would use Catalog mode with a GUI to show a list of formats. the list can be limited to specific criteria by using a filter. You can then select from the formats that are displayed. This is especially useful if you do not know which formats are available or the exact names of the formats.

Catalog Mode Internationalization

Catalog mode supports NDOs created containing catalog and catalog status information in any character encoding set. The default encoding for the XML documents created by the e-ADK is UTF-8 (Unicode). The user can select the encoding of the created XML documents by setting the configuration key Adapter.Serializer.Output.Encoding to the desired target encoding set. The user should be warned of two things. First, XML parsers tend to support a limited number of character encoding sets but almost universally support UTF-8. Make sure the target encoding set for the created XML documents is supported by the consuming XML parser. Second, the e-ADK does not throw an error when problems occur in transcoding data. If a character in the source NDO cannot be converted to the target encoding set, it will be replaced with a fill character.

Catalog Mode Example Configuration File

The following is an example configuration file with values set for Catalog mode:

```
### File Name: catalog.dat

Adapter
    adapter=NNADKStubPlugIn
    mode=CATALOG
    catalog.out=TestCatalog.xml
    catalog.out.status=CatalogStatus.xml
    #Optional key to control the encoding of the created
    #XML files
    Output.Serializer.Encording=Latin
```

Schema Remove

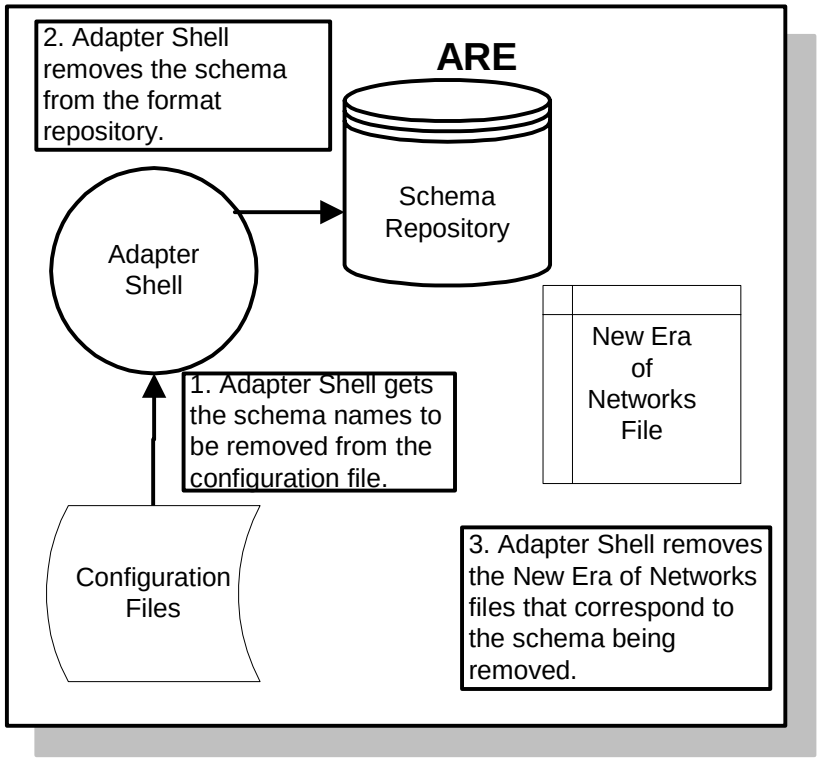
Schema Remove mode is used to remove format definitions that are stored in a repository. For the New Era of Networks Formatter schemas, this also removes any .ncm files. (An .ncm file is created each time you load a format to the formatter database. The .ncm file contains an abstraction of the format that was loaded to the formatter database.) Before calling deliverData() and processData(), the e-ADK verifies the incoming NDO against the format described by the XML in the .ncm file. If the NDO is not in the correct format, the e-ADK throws an exception, the message goes to the failure transport, and the next message is processed. If the structure of your data has changed, for example, if you change your SAP BAPI configuration, you should remove the old schema before installing the new schema.

The Adapter Shell requires Schema Remove mode in the configuration file to remove formats from the format repository. You must specify the appropriate SchemaLoader Factory and Library names. The names and valid entries are listed in “Configuration Keys” on page 48. The names of the top-level formats to be removed are specified in the schema.names configuration key. The remove.by.prefix must be set to false when you use the remove.schema.keys configuration key. Note that you must list all formats that you want removed using a comma-delimited list of format or schema names enclosed in parentheses. If you use the remove.by.prefix configuration key, you can remove all formats with the specified prefix by setting the key to true. Remember to comment out the actual values that are not used for the selected keys.

Schema Remove Model

The following diagram illustrates Schema Remove mode.

Figure 2-2: e-ADK Schema Remove Mode



Schema Remove Example Configuration File

The following is an example of a Schema Remove configuration file with values set to allow you to remove specific formats using the NNTSchemaLoader:

```
Adapter
    mode=SCHEMA_REMOVE
    repository.dir=c:\nnsy\NNSYContentRepository
    prefix=aaa
    test.drive=false
    session=ADKSession
    #To remove schemas by prefix
    #remove.by.prefix=true
```

```
    #remove.schema.keys=(N1,N2)
    #To remove specific schemas
    remove.by.prefix=false
    remove.schema.keys=(cc.IC.N1,cc.IC.N2,cc.OC.N1,
                        cc.OC.N2)
    SchemaLoader.Factory=NNTSchemaLoader_Factory
    #For all database types
    SchemaLoader.Library=adk39nnt56sl


Session.ADKSession
    #for MSSQL
    NNOT_SHARED_LIBRARY      =dbt26sql65
    NNOT_FACTORY_FUNCTION    =NNSesMS6Factory
    #for Oracle 8
    #NNOT_SHARED_LIBRARY      =dbt26or806
    #NNOT_FACTORY_FUNCTION    =NNSesOra8Factory
    #for DB2
    #NNOT_SHARED_LIBRARY      =dbt26db250
    #NNOT_FACTORY_FUNCTION    =NNSesDB2Factory
    #for Sybase
    #NNOT_SHARED_LIBRARY      =dbt26syb11
    #NNOT_FACTORY_FUNCTION    =NNSesSybCTFactory

    #as required by the database type
    NN_SES_SERVER            =test
    NN_SES_USER_ID           =testuser
    NN_SES_PASSWORD          =userpswd
    NN_SES_DB_NAME           =dbms56
```

Acquire

Acquire mode is used to get data from the application and put that data to a transport. The Adapter Shell running in Acquire mode calls an adapter plug-in. Acquire mode can be run with two different types of data: tree or buffer, each of which is described in more detail in the following sections.

When the user code processes the request, the return option is either TRUE or FALSE. TRUE tells the Adapter Shell to keep calling acquireData() or acquireBuffer(), and FALSE tells the Adapter Shell to stop calling acquireData() or acquireBuffer() to shutdown. After processing all of its requests, the Adapter Shell calls the shutdownAdapter() function, cleans up, and then exits.

Acquire Mode Internationalization

For Acquire NDO mode, the NDO data tree created by the adapter can be encoded in any valid character encoding set supported by core infrastructure version 2.1. The user can also select the encoding of the serialized message by setting the Adapter.Output.Serializer.Encoding key to the desired encoding in the configuration file. Default encoding of the serialized message is slightly different for the e-ADK supported serializers. For NCF messages, the default encoding of the serialized message is the native system encoding. For XML messages, the default encoding of the serialized message is UTF-8.

Warning!

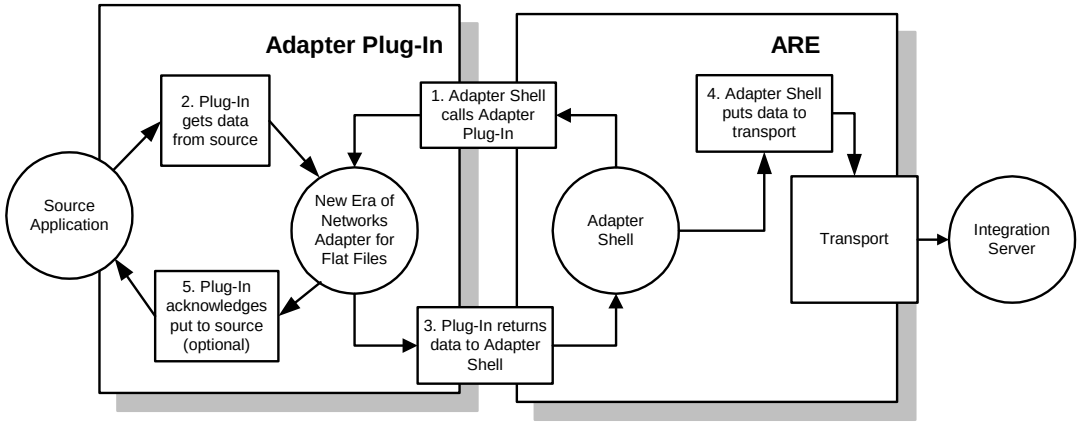
Transcoding errors do not throw or log exceptions. If a character in the source encoding does not exist in the target encoding, then it is replaced with a fill character.

XML parsers generally support a limited number of supported character encoding sets. Creation of an XML document in an encoding set that is unsupported by the target system will result in an illegible message although the XML is well formed.

Acquire Mode Model

The following diagram illustrates Acquire mode.

Figure 2-3: e-ADK Acquire Mode



Acquire Tree Mode

The acquireData() function is called to get data in an NDO tree from the application. An NDO object is constructed and is passed into the adapter plug-in. This plug-in should populate the NDO data tree and return an indication on whether the loop should continue or not. After the NDO has been populated, the data tree is extracted from the NDO and is put to the transport if the data tree is not empty.

If you are using an NDO with the NCF Serializer, the message type is used to look up the metadata in the schema repository. You can set the message type using the msg.type configuration key as detailed in “Configuration Keys” on page 48. OPT_MESSAGE_TYPE is more likely to be set dynamically in the adapter or by the NDO name. To keep the schemas cleaner when using the NCF wire format, not all schema information is kept in the wire format. If a message is being retrieved (Deliver mode) from a transport using NCF Serializer, the message must have the OPT_MSG_TYPE option set to the format name matching the schema repository file. When running MQSI or to place the message on the transport accessed by the deliver, the MQS_PROPAGATE option must be set to PROPAGATE in the rules GUI for the putqueue action placing the message on the transport. Otherwise, the message will not have the OPT_MSG_TYPE set.

Acquire NDO Mode Example Configuration File

The following is an example configuration file with values set for Acquire mode using NDO data representation:

```
### File Name: acquirendo.dat

Adapter
    adapter=nnadkstub
    mode=ACQUIRE
    data=NDO
    msg.type=NDO
    set.msg.options=true
    app.group=test
    transport.context.name=ADKContext
    transport.out.name=OUTQ
    transport.failure_store_name=FAIL
    maximum.transport.retries=2
    prefix=aaa
    set.oob.options=true
    acknowledge.put=true
    #NCF Serializer
    Output.Serializer.Factory=NCFSerializer_Factory
    Output.Serializer.Library=adk39ncfsd
    Output.Serializer.Encoding=Latin2
    #XML Serializer
    #Output.Serializer.Factory=XMLSerializer_Factory
    #Output.Serializer.Library=adk39xmlsd
    #Output.Serializer.Encoding=UTE8
    I18N_Condense=false

OTContext.ADKContext
    NNOT_CTX_DEFAULT_TIL_ID = FAIL
    NNOT_CTX_TMID           = MQSeriesTM
    NNOT_CTX_ENFORCE_TX     = TRUE

TransactionManager.MQSeriesTM
    NNOT_SHARED_LIBRARY    = oti26mqstm
    NNOT_FACTORY_FUNCTION = NOTMQSeriesTXManagerFactory
    NN_TM_MQS_QMGR         = TEST_QMGR

Session.ADKSession
    NNOT_SHARED_LIBRARY    = dbt26mqs
    NNOT_FACTORY_FUNCTION  = NNMQSSessionFactory
    NNMQS_SES_OPEN_QMGR    = TEST_QMGR

Transport.OUTQ
    NNOT_SHARED_LIBRARY        = dbt26mqs
```

```
    NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
    NNOT_TIL_OPEN_SESSION_ID   = ADKSession
    NNOT_TIL_OPEN_TSI          = TEST_OUT

Transport.FAIL
    NNOT_SHARED_LIBRARY        = dbt26mqs
    NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
    NNOT_TIL_OPEN_SESSION_ID   = ADKSession
    NNOT_TIL_OPEN_TSI          = TEST_FAIL
```

Acquire Buffer Mode

The acquireBuffer() function is used to get a buffer message from the application. The data buffer is constructed and passed into the adapter plug-in. This plug-in populates the buffer and returns an indication of whether the loop should continue. After the data buffer is populated, the buffer message is put to the transport.

Acquire Buffer Mode Example Configuration File

The following is an example configuration file with values set for Acquire mode using Buffer data representation:

```
### File Name:  acquire.dat
Adapter
    adapter=nnadkstub
    mode=ACQUIRE
    data=BUFFER
    msg.type=Buffer
    set.msg.options=true
    app.group=test
    transport.context.name=ADKContext
    transport.out.name=OUTQ
    transport.failure_store_name=FAIL
    maximum.transport.retries=2
    set.oob.options=true
    acknowledge.put=true

OTContext.ADKContext
    NNOT_CTX_DEFAULT_TIL_ID    = FAIL
    NNOT_CTX_TMID              = MQSeriesTM
    NNOT_CTX_ENFORCE_TX        = TRUE
```

```
TransactionManager.MQSeriesTM
    NNOT_SHARED_LIBRARY    = oti26mqstm
    NNOT_FACTORY_FUNCTION=NNOTMQSeriesTXManagerFactory
    NN_TM_MQS_QMGR         = TEST_QMGR

Session.ADKSession
    NNOT_SHARED_LIBRARY    = dbt26mqs
    NNOT_FACTORY_FUNCTION  = NNMQSSessionFactory
    NNMQS_SES_OPEN_QMGR    = TEST_QMGR

Transport.OUTQ
    NNOT_SHARED_LIBRARY        = dbt26mqs
    NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
    NNOT_TIL_OPEN_SESSION_ID   = ADKSession
    NNOT_TIL_OPEN_TSI          = TEST_OUT

Transport.FAIL
    NNOT_SHARED_LIBRARY        = dbt26mqs
    NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
    NNOT_TIL_OPEN_SESSION_ID   = ADKSession
    NNOT_TIL_OPEN_TSI          = TEST_FAIL
```

Deliver

Deliver mode is used to get information from the transport and deliver it to the application. The deliverData() function assumes that the data in the transport is a serialized data tree: either a serialized NCF format or a serialized XML format. The deliverBuffer() function receives raw data from the Adapter Shell as it existed on the transport. Deliver mode requires .ncm files to deserialize between the transport and the Adapter Shell.

When the user code processes the request, the return option is either TRUE or FALSE. TRUE tells the adapter plug-in to keep calling deliverData() or deliverBuffer() if data exists to process, and FALSE tells the adapter plug-in to stop calling deliverData() or deliverBuffer() and to shutdown. After processing all of its requests, the Adapter Shell calls the shutdownAdapter() function, cleans up, and then exits.

If the exit_if_empty key is set to TRUE and no message is found when the transport is read, the loop for Deliver mode terminates. If Deliver is retrieving a message from a transport using NCF Serializer, the message must have the OPT_MSG_TYPE option set to the format name matching the schema repository file.

If the adapter plug-in has an error, the Adapter Shell puts the message received to the failure transport. It then determines the severity of the error. For a critical error, the adapter shuts down. For a retry error, the adapter retries the specified number of times entered in the configuration file and then shuts down. Otherwise, processing continues with the next message. For additional information about critical and non-critical errors, see “Handling Exceptions” on page 76.

Batch Deliver is new functionality provided by the e-ADK 3.9. Batch Deliver allows the adapter to receive multiple messages before committing the messages as a single transaction. Batch Deliver functionality is selected by setting the configuration key Adapter.batch.size to an integer greater than 1. The e-ADK delivers the specified number of messages to the adapter before it calls the commitDeliverMessages() function in the adapter. The adapter returns one of three states to the e-ADK:

- Commit the messages and continue
- Rollback the messages and continue
- Rollback the messages and shut down

Based on the return parameter, the e-ADK responds accordingly.

The Adapter Shell running in Deliver mode calls an adapter plug-in. It can then deliver data to the application from the transport. Deliver mode can be run with two different types of data: tree (NDO) or buffer. If using an NDO, the message type is used to look up the metadata in the NDORepository. Not all schema information is kept in the wire format to keep the formats cleaner.

Batch Deliver mode has an additional configuration key: Adapter.batch.timeout. This key is the maximum number of times in seconds that a batch can process before initiating the commit function in the adapter. Setting the key to 0 results in no timeout for batches. The default setting is 0.

Deliver Mode Internationalization

For Deliver NDO mode, the e-ADK deserializes the message into an NDO. The default encoding of the created NDO is the native system encoding for all supported deserializers. The user can select the encoding of the created NDO by setting the Adapter.Input.Serializer.Encoding key to the desired character encoding set.

Warning!

Transcoding errors do not throw or log exceptions. If a character in the source encoding does not exist in the target encoding, then it is replaced with a fill character.

The e-ADK XML deserializer supports XML documents of a limited number of character encoding sets. Receipt of a message in an unsupported character encoding set will result in an illegible message although the created XML is well formed. The e-ADK XML deserializer uses the IBM XML4C version 4.0.0 parser. Supported character encoding sets include the following:

- ASCII
- UTF-8
- ISO-8859-1 (Latin 1)
- ISO-8859-2
- ISO-8859-3
- ISO-8859-4
- ISO-8859-5
- ISO-8859-6
- ISO-8859-7
- ISO-8859-8
- ISO-8859-9
- gb2312
- Big5
- koi8-r
- Shift_JIS

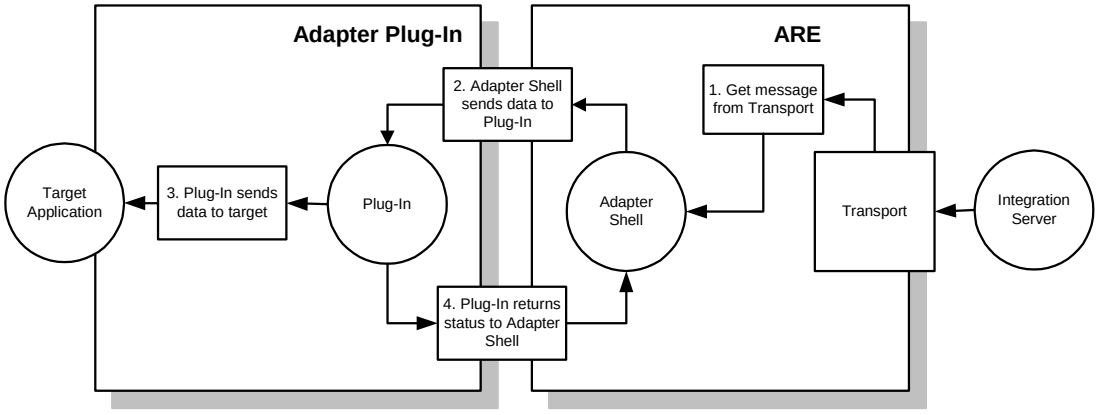
- euc-kr

Warning! EDCDIC and UTF-16 are not supported by the e-ADK XML deserializer. Using these character encoding sets will result in undetermined results.

Deliver Mode Model

The following diagram illustrates Deliver mode:

Figure 2-4: e-ADK Deliver Mode



Deliver Tree Mode

The deliverData() function gets data in the form of an NDO from a transport and delivers it to an application. The data tree is extracted from the NDO and delivered to the adapter plug-in call.

If you are using an NDO with the NCF Serializer, the message type is used to look up the metadata in the schema repository. To keep the formats cleaner when using the NCF wire format, not all schema information is kept in the wire format. If Deliver is retrieving a message from a transport using NCF Serializer, the message must have the OPT_MSG_TYPE option set to the format name matching the schema repository file.

Deliver NDO Mode Example Configuration File

The following is an example configuration file for Deliver mode using NDO data representation:

```
### File Name:  deliverndo.dat

Adapter
    adapter=nnadkstub
    mode=DELIVER
    data=NDO
    repository.dir=c:\nnsy\NNSYContentRepository
    prefix=aaa
    transport.context.name=ADKContext
    transport.in.name=INQ
    transport.failure_store_name=FAIL
    maximum.transport.retries=2
    transport.exit_if_empty=true
    #NCF Serializer
    Input.Serializer.Factory=NCFSerializer_Factory
    Input.Serializer.Library=adk39ncfsd
    #XML Serializer
    #Input.Serializer.Factory=XMLSerializer_Factory
    #Input.Serializer.Library=adk39xmlsd
    batch.size=0

OTContext.ADKContext
    NNOT_CTX_DEFAULT_TIL_ID    = FAIL
    NNOT_CTX_TMID              = MQSeriesTM
    NNOT_CTX_ENFORCE_TX        = TRUE

TransactionManager.MQSeriesTM
    NNOT_SHARED_LIBRARY = oti26mqstm
    NNOT_FACTORY_FUNCTION=NNOTMQSeriesTXManagerFactory
    NN_TM_MQS_QMGR         = TEST_QMGR

Session.ADKSession
    NNOT_SHARED_LIBRARY    = dbt26mqs
    NNOT_FACTORY_FUNCTION  = NNMQSSessionFactory
```

```
    NNMQS_SES_OPEN_QMGR    = TEST_QMGR

Transport.INQ
    NNOT_SHARED_LIBRARY        = dbt26mqs
    NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
    NNOT_TIL_OPEN_SESSION_ID   = ADKSession
    NNOT_TIL_OPEN_TSI          = TEST_IN

Transport.FAIL
    NNOT_SHARED_LIBRARY        = dbt26mqs
    NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
    NNOT_TIL_OPEN_SESSION_ID   = ADKSession
    NNOT_TIL_OPEN_TSI          = TEST_FAIL
```

Deliver Buffer Mode

The deliverBuffer() function gets data in the form of a buffer from a transport and delivers it to an application. The buffer is delivered to the adapter using a user-written plug-in call. The result of the user-written delivery should provide an indication of whether looping should continue.

Deliver Buffer Mode Example Configuration File

The following is an example configuration file with values set for Deliver mode using Buffer data representation:

```
### File Name:  deliver.dat

Adapter
    adapter=nnadkstub
    mode=DELIVER
    data=BUFFER
    transport.context.name=ADKContext
    transport.in.name=INQ
    transport.failure_store_name=FAIL
    maximum.transport.retries=2
    transport.exit_if_empty=true

OTContext.ADKContext
    NNOT_CTX_DEFAULT_TIL_ID    = FAIL
    NNOT_CTX_TMID              = MQSeriesTM
    NNOT_CTX_ENFORCE_TX        = TRUE

TransactionManager.MQSeriesTM
```

```
    NNOT_SHARED_LIBRARY    = oti26mqstm
    NNOT_FACTORY_FUNCTION=NNOTMQSeriesTXManagerFactory
    NN_TM_MQS_QMGR         = TEST_QMGR

Session.ADKSession
    NNOT_SHARED_LIBRARY    = dbt26mqs
    NNOT_FACTORY_FUNCTION  = NNMQSSessionFactory
    NNMQS_SES_OPEN_QMGR    = TEST_QMGR

Transport.INQ
    NNOT_SHARED_LIBRARY = dbt26mqs
    NNOT_FACTORY_FUNCTION = NNMQSQueueFactory
    NNOT_TIL_OPEN_SESSION_ID = ADKSession
    NNOT_TIL_OPEN_TSI = TEST_IN

Transport.FAIL
    NNOT_SHARED_LIBRARY        = dbt26mqs
    NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
    NNOT_TIL_OPEN_SESSION_ID   = ADKSession
    NNOT_TIL_OPEN_TSI          = TEST_FAIL
```

Process

Process mode is used to get data from a transport, enrich the data or submit a request and get a response, and put the data to another transport. Process mode is most appropriate when interfacing with applications that operate in a synchronous manner such as a CORBA call, an RFC request reply, or a database SELECT. The user must create a processBuffer() or a processData() function for the Process mode to perform enrichment on the incoming data or a request-reply mechanism.

In Process mode, the Adapter Shell takes information from a transport, calls your adapter plug-in library for processing, and delivers the resulting data to an output transport. Process mode runs with two types of data: tree (NDO) or buffer. The processData() function assumes that the data on the transport is a serialized data tree. The Adapter Shell deserializes the data, passes a tree to processData(), and then expects a modified or new data tree to be passed back. The Adapter Shell then serializes the modified data tree and places the data on the output transport. The processBuffer() function receives raw data from the e-ADK and returns raw data.

If the adapter plug-in throws an error, Process determines the severity of the error to determine whether the message should be retried. If the error allows retires, the adapter retries the specified number of times and then puts the message received to the failure transport. If the adapter receives a critical error, the transaction is rolled back and the adapter shuts down.

Process mode retrieves data from a transport, calls the user-written processing function, and puts data to a transport. This effectively provides a shell for processing data without having to understand transports. Process mode can be used with two different types of data: tree or buffer.

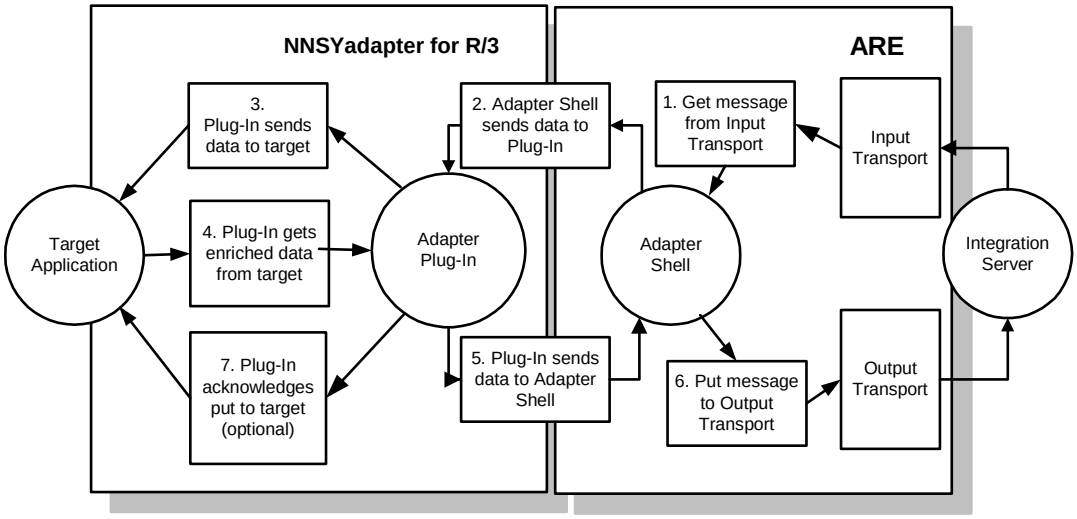
Process Mode Internationalization

Process NDO is effectively a Deliver NDO transaction and an Acquire NDO transaction linked together as a single transaction. Internationalization changes for the Deliver NDO transaction and the Acquire NDO transaction also apply to the Process NDO transaction.

Process Mode Model

The following diagram illustrates Process mode:

Figure 2-5: e-ADK Process Mode



Process Tree Mode

The Adapter Shell constructs two NDOs, one used during input and one used during output, and passes them to the processData() function. The data is retrieved from the transport into an NDO data tree, the adapter plug-in is called for processing, and the processing output is put to the transport. The result of the processing should provide an indication of whether looping should continue.

Process Tree Mode Example Configuration File

The following is an example configuration file with values set for Process mode using NDO data representation:

```
### File Name:  processndo.dat

Adapter
    adapter=nnadkstub
    mode=PROCESS
    data=NDO
```

```
    repository.dir=c:\nnsy\NNSYContentRepository
    prefix=aaa
    set.msg.options=true
    app.group=test
    transport.context.name=ADKContext
    transport.in.name=INQ
    transport.out.name=OUTQ
    transport.failure_store_name=FAIL
    maximum.transport.retries=2
    transport.exit_if_empty=true
    acknowledge.put=true
    #NCF Serializer
    Input.Serializer.Factory=NCFSerializer_Factory
    Input.Serializer.Library=adk39ncfsd
    Output.Serializer.Factory=NCFSerializer_Factory
    Output.Serializer.Library=adk39ncfsd
    #XML Serializer
    #Input.Serializer.Factory=XMLSerializer_Factory
    #Input.Serializer.Library=adk39xmlsd
    #Output.Serializer.Factory=XMLSerializer_Factory
    #Output.Serializer.Library=adk39xmlsd
    I18N_Condense=false

OTContext.ADKContext
    NNOT_CTX_DEFAULT_TIL_ID    = FAIL
    NNOT_CTX_TMID              = MQSeriesTM
    NNOT_CTX_ENFORCE_TX        = TRUE

TransactionManager.MQSeriesTM
    NNOT_SHARED_LIBRARY    = oti26mqstm
    NNOT_FACTORY_FUNCTION=NNOTMQSeriesTXManagerFactory
    NN_TM_MQS_QMGR         = TEST_QMGR

Session.ADKSession
    NNOT_SHARED_LIBRARY    = dbt26mqs
    NNOT_FACTORY_FUNCTION  = NNMQSSessionFactory
    NNMQS_SES_OPEN_QMGR    = TEST_QMGR

Transport.INQ
    NNOT_SHARED_LIBRARY        = dbt26mqs
    NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
    NNOT_TIL_OPEN_SESSION_ID   = ADKSession
    NNOT_TIL_OPEN_TSI          = TEST_OUT

Transport.OUTQ
    NNOT_SHARED_LIBRARY        = dbt26mqs
```

```
    NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
    NNOT_TIL_OPEN_SESSION_ID   = ADKSession
    NNOT_TIL_OPEN_TSI          = TEST_IN

Transport.FAIL
    NNOT_SHARED_LIBRARY        = dbt26mqs
    NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
    NNOT_TIL_OPEN_SESSION_ID   = ADKSession
    NNOT_TIL_OPEN_TSI          = TEST_FAIL
```

Process Buffer Mode

The Adapter Shell constructs two buffers, one used during input and one used during output, and passes them to the processBuffer() function. The data is retrieved from the transport into a buffer, the adapter plug-in is called for processing, and the processing output is put to the transport. The result of the processing should provide an indication of whether looping should continue.

Process Buffer Mode Example Configuration File

The following is an example configuration file with values set for Process mode using Buffer data representation:

```
### File Name:  process.dat

Adapter
    adapter=nnadkstub
    mode=PROCESS
    data=BUFFER
    msg.type=Buffer
    set.msg.options=true
    app.group=test
    transport.context.name=ADKContext
    transport.in.name=INQ
    transport.out.name=OUTQ
    transport.failure_store_name=FAIL
    maximum.transport.retries=2
    transport.exit_if_empty=true
    acknowledge.put=true

OTContext.ADKContext
    NNOT_CTX_DEFAULT_TIL_ID    = FAIL
    NNOT_CTX_TMID              = MQSeriesTM
```

```
    NNOT_CTX_ENFORCE_TX        = TRUE

TransactionManager.MQSeriesTM
    NNOT_SHARED_LIBRARY    = oti26mqstm
    NNOT_FACTORY_FUNCTION=NNOTMQSeriesTXManagerFactory
    NN_TM_MQS_QMGR         = TEST_QMGR

Session.ADKSession
    NNOT_SHARED_LIBRARY    = dbt26mqs
    NNOT_FACTORY_FUNCTION  = NNMQSSessionFactory
    NNMQS_SES_OPEN_QMGR    = TEST_QMGR

Transport.INQ
    NNOT_SHARED_LIBRARY        = dbt26mqs
    NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
    NNOT_TIL_OPEN_SESSION_ID   = ADKSession
    NNOT_TIL_OPEN_TSI          = TEST_OUT

Transport.OUTQ
    NNOT_SHARED_LIBRARY        = dbt26mqs
    NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
    NNOT_TIL_OPEN_SESSION_ID   = ADKSession
    NNOT_TIL_OPEN_TSI          = TEST_IN

Transport.FAIL
    NNOT_SHARED_LIBRARY        = dbt26mqs
    NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
    NNOT_TIL_OPEN_SESSION_ID   = ADKSession
    NNOT_TIL_OPEN_TSI          = TEST_FAIL
```

CHAPTER 3

Adapter Development Process

This chapter provides a checklist of prerequisites for developing an adapter. It also provides a broad outline of the steps for creating an adapter plug-in.

As you develop your adapter, consider this division of responsibilities and keep your adapter as focused as possible without including functionality that can be handled more effectively by other specialized servers.

Prerequisites

Before you begin the process of developing an adapter, be sure you have done the following:

- Install a supported integration server. See the *Installation and Configuration Guide* for the specific product.
- Install transport software. See the *Installation Guide* for a list of supported transports.
- Understand how to get your data from your application.
  - What is the data structure?
  - How do you want to present the data?
- Understand how data is sent to your specific application.
- Create the appropriate transports. If you are using transports to transport data to and from your adapter, you must create a transport manager and transports or a database instance (EMQ).

Example Files

The following files are available in the example directory:

- Examples of configuration files for each of the modes:
  - acquire.dat
  - acquirendo.dat
  - deliver.dat
  - deliverndo.dat
  - process.dat
  - processndo.dat
  - schema.dat
  - schema_remove.dat

- catalog.dat

Note You can rename the example configuration files with your own unique names. For consistency, the example configuration files in this guide use the .dat extension.

- The example makefile, which contains the standard options for both compilation and linking

  Example makefiles are provided to assist you with the development of an adapter. See the *e-ADK Programmer's Reference* for additional examples of these makefiles.

- NNADKStubPlugIn.cpp
- NNADKStubPlugIn.h

  Example NNADKStubPlugIn.cpp and NNADKStubPlugIn.h files are provided to add code for the functions. See "Modifying Adapter Functions" in the *e-ADK Programmer's Reference*.

The following methods are explained more fully in "Configuring the Environment" on page 46:

1 For NDO Mode:

- acknowledgePutData - Do any processing necessary as the result of the e-ADK notifying the adapter plug-in of the success or failure in putting a message to a transport. Applies to both Acquire and Process modes.
- handleQuietState - Do any processing necessary during a prolonged retry loop when the e-ADK is trying to get a message from a transport but the transport is empty. Applies to both Process and Deliver modes.

2 For Buffer Mode:

- acknowledgePutBuffer - Do any processing necessary as the result of the e-ADK notifying the adapter plug-in of the success or failure in putting a message to a transport. Applies to both Acquire and Process modes.
- handleQuietState - Do any processing necessary during a prolonged retry loop when the e-ADK is trying to get a message from a transport but the transport is empty. Applies to both Process and Deliver modes.

Developing an Adapter

When creating your adapter plug-in, your work will be divided between design time and runtime activities. “Defining Your Adapter Plug-In” on page 40 addresses the work during design time, and “Creating Configuration Files for Your Adapter Plug-In” on page 44 addresses the work during runtime.

Adapter Development Process

Assuming that you have a basic understanding of the Integration Server architecture and the ADK architectural model, begin the adapter development process following these steps:

1 Making design decisions:

- Identifying process and data models for a specific adapter
- Deciding which e-ADK modes to implement
- Deciding which data representation to use

2 Creating the Adapter plug-in.

3 Testing and implementing your adapter.

4 Deploying your adapter.

5 Maintaining and enhancing your adapter.

Defining Your Adapter Plug-In

During the design phase of your adapter, you need to decide which modes your adapter needs to handle. You also need to decide which type of data representation is best suited for your application if your adapter handles any or all of the following modes: Acquire, Process, and Deliver. See Chapter 2, “Modes” for descriptions of e-ADK modes.

Use the following criteria to decide how to handle the data from your application:

- Tree (NDO) method of data representation:
  - Data is already parsed.
  - You can use Schema mode to easily load a format into the formatter.

    - If the metadata changes in the end system, any format is easily recreated using Schema mode.
- Buffer method of data representation:
    - The adapter must contain the logic to parse the data buffer.
    - You must manually create formats to be used by the broker functionality of the integration server.
    - If the metadata changes in the end system, you must manually change the format.
- If the data is contiguous, for example, the segment size within an IDoc format is established by SAP. Segments do not have delimiters; segment size is the controlling factor. For that example, consider using a buffer to manage your data.
- If the data is noncontiguous, that is, the delimiters are used to separate segments, you should use an NNDOObject to manage your data in a tree-like structure.

You will need to implement the appropriate functions in your adapter to handle the modes and data representation that you have selected. For example, if you decide that your adapter should handle the acquire mode using buffered data, then you will need to implement acquireBuffer() inside your adapter. In addition, you need to implement the initAdapter() utility to remember configuration settings and allocate memory or data structure. The shutdownAdapter() functionality is used to clean up any allocated memory.

Finally, you may want to implement some special-purpose callback functions:

- acknowledgePutData()
- acknowledgePutBuffer()
- handleQuietState()
- commitDeliveredMessages()

Additional information about special-purpose callback functions is provided in *e-ADK Programmer's Reference*.

Creating Shared Libraries

The adapter developer must create a shared library or Dynamic Link Library (DLL) that contains functions to send or retrieve data from an application.

The e-ADK provides an example makefile to build the NNADKStubPlugIn.cpp into a library named:

- On NT, nnadkstub.dll
- On Solaris, libnnadkstub.so
- On HP-UX, libnnadkstub.sl
- On AIX, libnnadkstub.so

The functions must be named to conform to the e-ADK 3.9 requirements.

1 To set up your adapter, connect to your application, and create user data context:

   - initAdapter()

2 To shut down, disconnect from your application, and destroy user data:

   - shutdownAdapter()

3 For NDO Mode:

   - acquireSchema() - Create the structure to contain your data.
   - acquireData() - Retrieve data in an NDO from your application.
   - acquireCatalog() - Find formats available for your adapter.
   - deliverData() - Write data to your application.
   - acknowledgePutData() - Do processing necessary as the result of the e-ADK notifying the adapter plug-in of the success in putting a message (ack) or failing to put a message (nack) to a transport. Applicable for acquire and process modes.
   - processData() - Modify given data by interacting with your application.
   - handleQuietState() - Do processing necessary during a prolonged retry loop when the e-ADK is trying to get a message from a transport but the transport is empty. Applicable during Process and Deliver mode. For example, while waiting for messages from the transport, you could shut down your database or do some other processing to prevent your database from timing out. handleQuietState is supported in both tree and buffer data representation.

- commitDeliveredMessages() - Optional callback function that is required only if batch Deliver mode is implemented. The e-ADK calls this function to notify the adapter plug-in that it is ready to commit the delivered batch. Based on the return value, the e-ADK commits the batch and continues, rolls back the batch and continues, or rolls back the batch and initiates a shutdown.

4 For Buffer Mode:

- acquireBuffer() - Retrieve data in a buffer from your application
- deliverBuffer() - Write buffer data to your application
- acknowledgePutBuffer() - Do any processing necessary as the result of the e-ADK notifying the adapter plug-in of the success or failure in putting a message to a transport. Applicable for acquire and process modes.
- process buffer() - Modify given buffer data by interacting with your application
- commitDeliveredMessages() - Optional callback function that is required only if batch Deliver mode is implemented. The e-ADK calls this function to notify the adapter plug-in that it is ready to commit the delivered batch. Based on the return value, the e-ADK commits the batch and continues, rolls back the batch and continues, or rolls back the batch and initiates a shutdown.

The following classes are used by the adapter plug-in functions:

- NDO classes with header file references
  - include/NDO/NNDOObject.h
  - include/NDO/NNDODataTree.h
  - include/NDO/NNDOSchemaTree.h
  - include/NDO/NNDODataNode.h
  - include/NDO/NNDOSchemaNode.h
- include/ADK/NNDataBuffer.h
- include/ADK/NNADKOutOfBand.h
- include/CFG/NNConfig.h

Creating Configuration Files for Your Adapter Plug-In

After your adapter has been written, compiled, and linked, you need to create the configuration files that will be used in conjunction with NNSYAdapter39.exe. You create a unique configuration file for each mode and data representation that your adapter supports. For example, if your adapter implements the acquire mode for both NDO and buffer, you need two different configuration files, each one configured for the mode and data representation appropriate for that mode. When you run the adapter, you will need to set up one instance of the adapter for each configuration file.

At runtime, the functions inside your adapter are called at the appropriate time, depending on the mode that is running and, when the mode is Acquire, Process, or Deliver, the data representation that is active.

CHAPTER 4

Adapter Runtime Environment

The Adapter Runtime Environment (ARE) is required for testing and running the adapters. A runtime environment installation is provided so that only files needed by the runtime environment are provided. Therefore, when the ARE is delivered with the adapter you have created, it does not contain the executables and header files that are used to create the adapter.

Instructions for installation of the ARE are available in “Redistributing the Adapter Runtime Environment” on page 74.

This chapter provides the following information:

Adapter Shell Settings for Servers

Specific combinations of a server, OT driver, schema loader, and serializer are required for the runtime environment of e-ADK.

Table 4-1: Settings for Integration Servers

Server	Open Transport Driver	Schema Loader	Serializer
MQSI 2.0.2 WMQI 2.1	MQSeries (use RFH2)	NNT56 SchemaLoader	NCFSerializer
e-Biz Integrator 3.6	EMQ MQSeries MSMQ File	NNT56 SchemaLoader	NCFSerializer
Application Servers: EAServer WebSphere Application Server WebLogic Application Server	 JMS OT Driver MQSeries JMS OT Driver	 DTD DTD DTD	 NDOXML NDOXML NDOXML

Note If you are using multi-byte data, you must use RFH2 headers.

Configuring the Environment

The New Era of Networks configuration architecture comprises configuration items that are stored as read-only after initialization. The available configuration items are searched in the following order:

- Command-line arguments
- Configuration file (using grouping)
- Registry file (Windows NT only)
- Environment variables
- Defaults holder

Common File Format

Example configuration files are included in the examples>ADK directory. The examples illustrate the various modes and data representations. In the sample configuration files, the configuration file names have a .dat extension.

New Era of Networks adapters require the following common configuration files, which are structured using the following format. This format is an example of the structure you would use:

```
<group name or stanza name>
    <key 1>=<value1>
     <key 2>=<value2>
     <key n>=<valuen>
```

For example:

```
#sample configuration file
Adapter
    adapter=pluginname
     mode=ACQUIRE
     data=NDO
     queue.in.name=INQ
```

The keys are case sensitive.

The following keys must be considered as pairs:

- schema loader factory and schema loader library

  The schema loader library key contains the name of the schema loader library with the factory function. The library name must match the factory name used in Schema mode.

- input serializer factory and input serializer library
- output serializer factory and output serializer library
- input serializer encoding and output serializer encoding

Additional information about the use of these keys is available in “Configuration Keys” on page 48.

Key and value pairs are defined in the configuration file used during the startup of the e-ADK AdapterShell. Available keys are described in “Configuration Keys” on page 48.

Note The following warning message appears when the program starts up:

```
Default config file ’/Dir/Path/nnsyreg.dat’ is not
```

```
being used.
```

This warning message appears because the e-ARE does not use the nnsyreg.dat. This message can be ignored.

Configuration Keys

Configuration keys comprise a section and a key that points to a value.

Many of the configuration keys are optional. Whether you use them depends on the mode, data representation, or other specific functionality such as implementation of retries. The available keys and values are described in the following tables:

Standard Keys

Table 4-2: e-ADK Standard Keys and Values

Name	Values	Description
adapter	User-defined	The plug-in shared library name that the Adapter Shell should load and execute. Do not include the extension (.so, .sl, or .dll) or the lib prefix (Unix). The value for this key is user-defined and applies to all modes.
app.group	User-defined	Sets OPT_APP_GRP option. The value for this key is user-defined and applies to Acquire and Process modes.
batch.size	Integer Default is 1	Key used for batch Deliver mode. A value of greater than 1 signifies that batch mode has been selected. e-ADK will process the number of messages specified before calling commit. commitDeliveredMessages() function must be defined in the adapter if greater than one. This key can be used in combination with batch.size to be sure messages are not uncommitted for long periods. This key applies to Deliver mode.

Name	Values	Description
batch.timeout	Integer Default is 0	Key used for batch Deliver mode. Key is effective only when using Deliver mode and key batch.size is set to greater than 1. The value signifies the maximum time in seconds for a batch to process. If the elapsed time in seconds required to process a batch exceeds this key value, then the e-ADK initiates the process to commit that portion of the batch that has been completed. A value set to zero signifies no batch timeout period. This key can be used in combination with batch.timeout to be sure messages are not uncommitted for long periods. This key applies to Deliver mode.
catalog.out	User-defined	Defines the name of the target files in which to store the catalog XML document.
catalog.outstatus	User-defined	Defines the name of the target files in which to store the catalog status XML document.
data	Buffer NDO	Sets the data processing type: NDO (for tree) or Buffer. Schema mode handles only NDO data and Schema Remove ignores data. Both NDO and buffer data can be used in all other modes.
mode	ACQUIRE PROCESS DELIVER SCHEMA SCHEMA_REMOVE CATALOG	Sets the mode of use. Valid entries are Acquire, Catalog, Deliver, Process, Schema, or Schema_Remove.
msg.type	User-defined	When set.msg.options is set as true: sets OPT_MSG_TYPE option for buffer in Acquire and Process modes.

Name	Values	Description
set.msg.options	true false (default)	Attaches message options to the header when turned on. Sets message options OPT_APP_GRP and OPT_MSG_TYPE. Depends on APP.GROUP and MSG.TYPE set for BUFFER. Values are true or false. Default is false.
test.drive	true false (default)	Designed for testing an adapter plug-in only in Schema mode. Test Drive does not actually write to the format repository. The data is placed in the log as if trace were turned on. Used in Schema mode.

Transport Keys

A transport session is a configuration section containing data common to all of the configured transports. The configuration information is common to the transport connection, which is also known as session.

Table 4-3: e-ADK Transport Keys and Values

Name	Values	Description
transport.context.name	User-defined	Identifies context entry in the Open Transport Directory. The Open Transport Server configures the context object based on this entry. See Open Transport documentation for more information on this adapter key. Used in Acquire, Deliver, and Process modes.
transport.in.name	User-defined	Sets the logical transport input tag. Used in Deliver and Process modes.
transport.out.name	User-defined	Sets the logical transport output tag. Used in Acquire and Process modes.

Name	Values	Description
transport.failure_store_name	User-defined	Specifies the transport used for messages that were read from the input transport but failed the adapter plug-in. If no value is specified, the Adapter Shell will not run; it must be set for a transport. Used in Deliver, Process, and Acquire modes.
transport.exit_if_empty	true (default) false	When looping through the read of the transport, the deliver loop terminates if the read does not return with a message. Valid values are true or false. Default is true. Used in Process and Deliver modes for both Buffer and NDO data representation.

Name	Values	Description
target.wait_time	Integer	Empty transport wait time - in milliseconds. Value is used as a polling interval that starts at the end of message processing. Values used: • If there is no NNOT_TIL_GET_BLOCKING_TIMEOUT value, this value is used as a polling timeout. • If there is an NNOT_TIL_GET_BLOCKING_TIMEOUT value, this value is used in conjunction with the blocking timeout. For example, the blocking get applies first, then a sleep before another blocking get is tried. • If this value is not set, then only the NNOT_TIL_GET_BLOCKING_TIMEOUT value is used for blocking. Valid values are numeric. It is suggested that the user not set this to zero (0) milliseconds. Default is 1. Used in Deliver and Process modes.
adapter_wait_time	Integer	Length of time in milliseconds that the AdapterShell waits to retry a call to the Adapter after a retry exception is thrown. Valid values are numeric. Default is zero (0). Used in Deliver and Process modes.

Name	Values	Description
maximum.transport.retries	Integer	Maximum number of times the e-ARE attempts to put to or get a message from a transport after an exception occurs. For Deliver and Process, setting this value to zero (0) results in an infinite retry loop. Valid values are numeric. Default is zero (0). Used in Acquire, Deliver, and Process modes.
maximum.adapter.retries	Integer	Maximum number of times a call to an adapter is retried when a retry exception occurs. Valid values are numeric. Default is zero (0). Used in Acquire, Deliver, and Process modes.
enter.quiet.state	Integer	Indicates the retry iteration when the Adapter's handleQuietState() is called. This value must be less than adapter.maximum.transport.retries or the Adapter's handleQuietState() is never called. The key maximum.transport.retries sets up the number of times the transport call is retried when an exception occurs. The handle quiet functionality is designed to do adapter-specific processing inside the retry loop. If no retry loop exists, then the handle quiet function will not be called. For example, if maximum.transport.retries = 5 and enter.quiet.state = 2, then the handleQuietState() function is called every other time through the transport retry loop. If enter.quiet.state = 0, the handleQuietState() function is not called. Valid values are numeric. Used in Deliver and Process modes.

Name	Values	Description
acknowledge.put	true false (default)	Determines whether the adapter has a callback function that enables it to receive an ack or nack regarding a put to an output transport. If set to true, the adapter receives an ack or nack. Valid values are true or false. Default is false. Used in Acquire and Process modes.

Open Transport Keys

You must set the open transport keys and values. The available keys, values, and a description are provided in the open transport configuration guide for each specific OT driver.

Schema Loader Plug-Ins

Table 4-4: Schema Loader Required Keys

Name	Values	Description
SchemaLoader. Factory	NCMSchemaLoader_ Factory DTDSchemaLoader_ Factory XMLSchemaLoader_Factory NNTSchemaLoader_Factory	Indicates name of the factory function to use to create the SchemaLoader class. Must be paired with Schema Loader.Library. Used in Schema mode.
SchemaLoader. Library	adk39ncmsl adk39dtdsl adk39xmlsl adk39nnt56sl	Indicates the name of the Schema Loader library that contains the factory function. The library name must match the factory name and cannot contain a file extension or standard library naming. Must be paired with SchemaLoader.Factory. Used in Schema mode.

Name	Values	Description
prefix	User defined - ASCII values only	Adds prefix to the format name in Schema, Acquire, and Process in NDO mode. The message type name becomes <prefix> + "IC." + <NDO name>. (The IC [inbound compound shown here] or OC [outbound compound] is inbound to or outbound from the Formatter.) Prefix has a 5-character limit of ASCII characters. Used in Schema Remove, Schema, Acquire, and Process modes with NDO.
remove.by.prefix	true false (default)	Used to remove formats based on the prefix. Prefix has a 5-character limit. Used in Schema Remove mode.
remove.schema.keys	User-defined	List used to identify the formats to be removed. When remove by prefix is turned on, the list contains the format without prefix. When remove by prefix is not turned on, the list contains the full name of the formats to be removed. Specify formats as comma delimited with parenthesis. Used in Schema Remove mode.

NNTSchemaLoader

The NNTSchemaLoader should be used for testing the adapter or generating NCM files for the following Integration Servers:

- MQSI 2.0.2
- WMQI 2.1
- e-Biz Integrator 3.6
- New Era of Networks Rules and Formatter 5.6

NCF Serialization should be used for the runtime when this schema loader is used in design time.

Table 4-5: Keys for NNT56SchemaLoader

Name	Values	Description
clash.avoid	true false (default)	This switch allows a unique format name to be generated when a format is loaded into the Formatter database and that format name currently exists. Does not apply to a top-level format.
continue.format.exists	true false (default)	Allows user to specify whether the program sends out an error and exits (if set to false) when a format exists in the database. If set to true, it sends a warning message and continues loading remaining formats. Used in Schema mode.
I18N_Condense	true false (default)	The key designates use of specific methodology of condensing format names. False = Signals the e-ADK to use the old methodology of condensing the format name if necessary. The old methodology does not support multi-byte characters. True = Signals the e-ADK to use the new methodology of condensing the format name if necessary. The new methodology is a simple character truncation routine and supports multi-byte data. Used in Schema mode. It is also used in Acquire and Process modes for NDO data representation when the NCF serializer is used. **Warning!** Setting the key to false and using multi-byte format names will generate undetermined results.

Name	Values	Description
NCF.version	101 (default) 102	Indicates the New Era of Networks Canonical Format version to use when loading Schema.
prefix	User-defined - ASCII values only	Adds prefix to the format name. The message type name becomes <prefix>.IC.<NDO name>. Prefix has a 5-character limit of ASCII characters. Used in Schema, Acquire, and Process modes with NDO data representation.
repository.dir	User-defined. If no value specified, defaults to NNSYContentRepository	Indicates the directory in which NNSY content model files are written (.ncm files). Used in Schema, Deliver, and Process modes for NDO data representations.
schema.input	true (default) false	Set to true to generate input formats. (Input designates into broker.) Used in Schema mode.
schema.output	true (default) false	Set to true to generate output formats. (Output designates out of broker.) Used in Schema mode.
session	User-defined	User-defined key used to describe the OT session name. Used in Schema and Schema Remove mode.
remove.by.prefix	true false (default)	Used to remove formats based on the prefix. Prefix has a 5-character limit. Used in Schema Remove mode.
remove.schema.keys	User-defined	List used to identify the formats to be removed. When remove by prefix is turned on, the list contains the format without prefix. When remove by prefix is not turned on, the list contains the full name of the formats to be removed. Specify formats as comma delimited with parenthesis. Used in Schema Remove mode.

NCMSchemaLoader

The NCMSchemaLoader should be used for testing the adapter or generating NCM files for the following Integration Servers:

- MQSI 2.02
- WMQI 2.1
- e-Biz Integrator 3.6
- New Era of Networks Rules and Formatter 5.6

NCF Serialization should be used for the runtime when this schema loader is used in design time.

Table 4-6: Keys for NCMSchemaLoader

Name	Values	Description
clash.avoid	true false (default)	This switch allows a unique format name to be generated when the Formatter loads a format into the database and that format name currently exists. Does not apply to top-level formats.
continue.format.exists	false (default) true	Allows user to specify whether the program sends out an error and exits if an ncm already exists for the format. If set to true and the ncm exists, the program sends a warning message and continues creating ncms. If set to false and the format exists, the program logs an error and shuts down. Used in Schema mode.
prefix	User-defined - ASCII values only	Adds prefix to the format name. The message type name becomes <prefix>IC.<NDO name>. Prefix has a 5-character limit. Used in Schema, Acquire, and Process modes for NDO data representations.
repository.dir	User-defined. If no value specified, defaults to NNSY_ROOT/NNSYContentRepository	Indicates the directory in which NNSY content model files are written (.ncm files). Used in Schema, Deliver, and Process modes for NDO data representations.

Name	Values	Description
schema.input	true (default) false	Set to true to generate input formats. Used in Schema mode.
schema.key	User-defined	Used to retrieve schema from adapter plug-in and to retrieve schema from NCM repository.
schema.output	true (default) false	Set to true to generate output formats. Used in Schema mode.
session	User-defined	Used to describe the OT session name. Used in Schema and Schema Remove mode.

DTDSchemaLoader

The DTDSchemaLoader should be used for testing the adapter or generating a DTD for the following Integration Servers:

- MQSI 2.0.2
- WMQI 2.1
- e-Biz Integrator 3.6
- New Era of Networks Rules and Formatter 5.6

It can also be used with the following Application Servers:

- EAServer
- WebSphere Application Server
- WebLogic Application Server

XML Serialization should be used for the runtime when this schema loader is used in design time. This schema loader can be used best if the New Era of Networks Formatter will not be used during runtime.

Table 4-7: Keys for DTDSchemaLoader

Name	Values	Description
continue.format.exists	false (default) true	Allows user to specify whether the program sends out an error and exits if a DTD already exists for the format. If set to true and the DTD exists, the program sends a warning message and continues creating DTDs. If set to false and the DTD exists, the program logs an error and shuts down Used in Schema mode.
repository.dir	User-defined. If no value specified, defaults to NNSY_ROOT/NNSYContentRepository	Indicates the directory in which NNSY content model files are written (.ncm files). Used in Schema, Deliver, and Process modes for NDO data representations.

XMLSchemaLoader

The XMLSchemaLoader should be used for testing the adapter or generating an XML Schema file.

- MQSI 2.0.2
- WMQI 2.1
- e-Biz Integrator 3.6
- New Era of Networks Rules and Formatter 5.6

XML Serialization should be used for the runtime when this schema loader is used in design time. This schema loader can be used best if the New Era of Networks Formatter will not be used during runtime.

Table 4-8: Keys for XMLSchemaLoader

Name	Values	Description
continue.format.exists	false (default) true	Allows user to specify whether the program sends out an error and exits if an XML already exists for the format. If set to true and the XML exists, the program sends a warning message and continues creating XMLs. If set to false and the XML exists, the program logs an error and shuts down Used in Schema mode.
repository.dir	User-defined. If no value specified, defaults to NNSY_ROOT/NNSYContentRepository	Indicates the directory in which NNSY content model files are written (.ncm files). Used in Schema, Deliver, and Process modes for NDO data representations.

Serialization/Deserialization Plug-Ins

Serialization is the process of walking a tree and writing the data for each node to wireform. This generates a wireform containing the format representation of the data in the tree. It then converts the wireform or contiguous messages so they can be used for transport or persistence.

Deserialization is the process of converting messages from wireform or contiguous formats and creating the corresponding tree representation.

Serialization and deserialization plug-ins are used to convert message data so that it can be used for transport, persistence, or by another program. The serializer plug-ins are used during runtime paired with schema loaders used during design time.

The e-ARE allows you to determine which format to use when putting data from an NNDOObject DataTree to a transport. The format is chosen based on four configuration keys when running the adapter.

The Input.Serializer keys are used in Deliver and Process modes (NDO data only) to deserialize from the wireform received from the input transport into a NNDOObject to be accepted by the adapter plug-in.

- Input.Serializer

  Factory: NCFSerializer_Factory

  Library: adk39ncfsd

  Modes: Deliver and Process

  Action: Deserialize

  From: data in wireform received from input transport

  To: NNDOObject to be sent to adapter plug-in

  Handles Formats: NCF 101 (New Era of Networks Canonical Format, version 101) and NCF 102

  Integration Server: MQSI 2.0.2, WMQI 2.1, e-Biz Integrator 3.6

- Input.Serializer

  Factory: XMLSerializer_Factory

  Library: adk39xmlsd

  Modes: Deliver and Process

  Action: Deserialize

  From: Data in wireform received from input transport

To: NNDOObject to be sent to adapter plug-in

Handles Formats: XML

Integration Servers: MQSI 2.0.2, WMQI 2.1, e-Biz 3.6,

Application Servers: EAServer, WebSphere Application Server, and WebLogic Application Server

The Output.Serializer keys are used in Acquire and Process modes (NDO data only) to serialize the NNDOObject received from the adapter plug-in to be placed on the output transport.

- Output.Serializer

  Factory: NCFSerializer_Factory

  Library: adk39ncfsd

  Modes: Acquire and Process

  Action: Serialize

  From: NNDOObject received from adapter plug-in

  To: Data in wireform to be placed on outside transport

  Handles Formats: NCF 101 (New Era of Networks Canonical Format, version 101) and NCF 102

  Integration Servers: MQSI 2.0.2, WMQI 2.1, and e-Biz Integrator 3.6

  Application Servers: EAServer, WebSphere Application Server, and WebLogic Application Server

- Output.Serializer

  Factory: XMLSerializer_Factory

  Library: adk39xmlsd

  Modes: Acquire and Process

  Action: Serialize

  From: NNDOObject received from adapter plug-in

  To: data in wireform to be placed on outside transport

  Handles Formats: XML

  Integration Server: MQSI 2.0.2 and WMQI 2.1

Table 4-9: Serializer Required Keys

Name	Values	Description
Input.Serializer.Factory	NCFSerializer_Factory XMLSerializer_Factory	Indicates the factory function used to create the Serializer/Deserialzer class. This Serializer deserializes the message data into an NDO for the adapter to use. Used in Deliver and Process mode for NDO data representation.
Input.Serializer.Library	adk39ncfsd adk39xmlsd	Indicates the name of the Serializer library that contains the factory function. The library name and factory name must be associated; for example, both contain XMLSerializer or both contain NCFSerializer. The names cannot contain a file extension or standard library naming. Used in Deliver and Process modes for NDO data representation.
Input.Serializer.Encoding	User-defined	Used to specify the encoding section of the NDO created by the deserialization process. If no encoding is set, the system defaults to the native system encoding. Used in Acquire and Process modes for NDO data representation.
Output.Serializer.Factory	NCFSerializer_Factory XMLSerializer_Factory	Indicates the name of the factory function to use to create the Serializer/Deserializer class. This Serializer is used to serialize the NDO returned from the adapter into message data. Used in Acquire and Process modes for NDO data representation.

Name	Values	Description
Output.Serializer.Library	adk39ncfsd adk39xmlsd	Indicates the name of the Serializer library containing the factory function. The library name and factory name must be associated; for example, both contain XMLSerializer or both contain NCFSerializer. the name cannot contain a file extension or standard library naming. Used in Acquire and Process modes for NDO data representation.
prefix	User defined - ASCII values only	Adds prefix to the format name in Schema, Acquire, and Process in NDO mode. The message type name becomes <prefix> + “IC.” + <NDO name>. Prefix has a 5-character limit. Used in Schema Remove, Schema, Acquire, and Process modes with NDO.
Output.Serializer.Encoding	User-defined	Used to specify the encoding set of the serialized message. If no encoding set is specified, the system defaults to the native system encoding for NCM or UTF-8 for XML. Used for Deliver and Process modes for NDO data representation.

NCF Serializer

The NCF Serializer puts the data out in an ncm stream of data. The NCFSerializer should be used for the following Integration Servers:

- MQSI 2.0.2
- WMQI 2.1
- e-Biz Integrator 3.6
- New Era of Networks Rules and Formatter 5.6

The NNTSchemaLoader should have been used for the schema loading when this format is used.

Table 4-10: Keys for NCFSerializer

Name	Values	Description
I18N_Condense	true false (default)	This key designates use of specific methodology of condensing format names. False = Signals the e-ADK to use the old methodology of condensing the format name if necessary. The old methodology does not support multi-byte characters. True = Signals the e-ADK to use the new methodology of condensing the format name if necessary. The new methodology is a simple character truncation routine and supports multi-byte data. Used in Schema mode. It is also used in Acquire and Process modes for NDO data representation when the NCF serializer is used. **Warning!** Setting the key to false and using multi-byte format names will generate undetermined results.
NCF.version	101 (default) 102	Indicates the New Era of Networks Canonical Format version to use when loading schema. Used in Acquire and Process modes.

Name	Values	Description
prefix	User-defined - ASCII values only	Adds prefix to the format name. The message type name becomes <prefix>IC.<NDO name>. Prefix has a 5-character limit. Used in Schema, Acquire, and Process modes for NDO data representations. Used in Acquire and Process mode for NCFSerializer.
repository.dir	User-defined. If no value specified, defaults to NNSY_ROOT/NNSYContentRepository	Indicates the directory in which NNSY content model files are written (.ncm files). Used in Schema, Deliver, and Process modes for NDO data representations.
set.msg.options	true false (default)	Sets the following message options, depending on APP_GROUP and MSG_TYPE set for BUFFER: OPT_APP_GROUP OPT_MSG_TYPE Used in Acquire and Process modes.

XMLSerializer

This serializer puts the data out in a XML Stream of data.

The XMLSerializer should be used for the following Integration Servers:

- MQSI 2.0.2
- WMQI 2.1

The XMLSerializer should be used for the following Application Servers:

- EAServer
- WebSphere Application Server
- WebLogic Application Server

The XMLSchemaLoader or DTDSchemaLoader should have been used for the schema loading when this format is used.

Table 4-11: Keys for XMLSerializer

Name	Values	Description
prefix	User-defined - ASCII values only	Used in Schema, Acquire, and Process modes for NDO data representations. Prefix has a 5-character limit. Adds prefix to the format name. The message type name becomes <prefix>IC.<NDO name>.
repository.dir	User-defined. If no value specified, defaults to NNSY_ROOT/NNSYContentRepository	Indicates the directory in which NNSY content model files are written (.ncm files). Used in Schema, Deliver, and Process modes for NDO data representations.
set.msg.options	true false (default)	Used in Acquire and Process modes. Sets the following message options, depending on APP_GROUP and MSG_TYPE set for BUFFER: OPT_APP_GROUP OPT_MSG_TYPE

Encrypting the Configuration File

The NNCrypt utility allows you to encrypt and decrypt the configuration file to protect sensitive information, such as usernames and passwords. Run the NNCrypt utility against your configuration file to encrypt or decrypt it.

Syntax

```
NNCrypt (-encrypt | -decrypt) -file
```

Table 4-12: Parameters for Encryption

Parameter	Values	Description
-encrypt		Encrypts the specified file. Returns an error if the file is already encrypted.
-decrypt		Decrypts the specified file. Returns an error if the file is already decrypted.
-file		Specifies the name of the file to encrypt or decrypt.

Encrypted files do not use a .crypt extension or take priority over decrypted files as the program searches for configuration information.

Testing

e-ADK 3.9 provides several testing methods for use during runtime. These methods allow the developer to view the intended results of the adapter without actually updating databases or adding messages to transports. You set up testing differently, depending on the mode you are using:

- For Schema and Schema Remove, set up test drive in the configuration files
- For Acquire, Process, and Deliver modes, set up and use the OT file driver in the configuration file

Using Test Drive in Schema Mode

You can use test drive to test a schema before it is actually created. You do not have to be connected to the database to view the schema. No *.ncm files are created during test drive.

❖ **To set up test drive for Schema mode**

1 Open the schema configuration file.

2 Add the following to the schema configuration file Adapter.test.drive=true.

The schema tree displays. You can review this schema before it is entered into the repository.

Using OT File Driver for Testing

The OT driver is used for testing during Acquire, Process, and Deliver modes. Use the OT driver to place messages in files rather than to a transport. Set up the OT file driver to enable this function for each of Acquire, Process and Deliver modes. Sample configuration files that illustrate how to set up a file driver are provided in the examples directory as acquirefile.dat, processfile.dat, and deliverfile.dat.

Warning! If test.drive=true in Acquire, Process, or Deliver mode configuration files, you will receive the critical error message:

Test Drive unavailable. Set test.drive=false. Use the file driver instead.

Executing NNSYAdapter39

You can execute the adapter from the command line or, if you are using Windows, from the control panel. To control the launch from the control panel, you must register your adapter as an NT Service.

Executing the Adapter Shell from the Command Line

To execute the Adapter Shell from the command line, follow the format of the examples below. If no command line options are used, the code assumes the executable will read the configuration from NNSYAdapter.cfg in the local directory or, if that is not found, nnsyreg.dat in the [NNSY_ROOT] directory. The following is the typical command line. Other parameters may be listed if they are not in the configuration file or if the value in the configuration file needs to be overwritten.

Example: NNSYAdapter39 -file = <filename> [-trace]

Note Invoking your Adapter for each mode involves the same command line procedure. Only the configuration files change for each mode.

Registering the Adapter as an NT Service

You can control the launch and other functions of your adapter from the control panel if you register your adapter as an NT Service.

Prerequisites

- Verify that all dependent DLLs are loaded into a directory that is in the Windows NT system path.
- If the NT service needs access to network resources such as DLLs on a mapped network drive or MSMQ transports on the network, you must connect to the network drives after you install NT service but before starting the service.

Connecting to the Network

1 Go to Start→Settings→Control Panel.

2 Click on Services.

3 Double click on the service that was installed.

4 From the Logon As group box, select This Account.

5 Type in your information in the following fields:

- Domain Name\User Name
- User Password
- Confirm User Password

Using the Services Panel

Services can be set up using different methods. The following details one of the methods for installing a service.

1 To install a service, at the command line, type:

NNSYAdapter39 -*install* [-n|-name] serviceName [-d|-display] serviceDisplayName -p paramFile

2 To remove a service, at the command line, type:

NNSYAdapter39 -*remove* [-n|-name] serviceName

3 To start a service, at the command line, type:

NNSYAdapter39 -*start* [-n|-name] serviceName

4 To stop a service, at the command line, type:

NNSYAdapter39 -*stop* [-n|-name] serviceName

5 To see the current status of a service, at the command line, type:

NNSYAdapter39 -*status* [-n|-name] serviceName

You can check your NT service information using the Services Panel. The Services Panel is invoked from Start→Control Panel→Services where you can review the list.

Examples

Each service entry has one name for the service and one for the displayed name. These two names do not have to be the same but you can have only one service for each name. If you install a named service, you need to use that same name when removing it.

The default name for NNSYAdapter39 is NNSYADK. If you want more than one NNSYAdapter39 service, you name them using the -install -n[ame] <name> convention. The display name is the name that appears in the list of services that you can review from Control Panel Services.

- The following installs a service with the name adk39Svc and uses the configuration file NNadk39.dat.

  NNSYAdapter39 -install -n adk39Svc -p NNadk39.dat

- The following installs a service with the name adk39Svc with no parameters:

  NNSYAdapter39 -install -n NNadk39Svc

- The following removes a service named NNadk39:

  NNSYAdapter39 -remove -n NNadk39

- The following starts a service named NNadk39:

  NNSYAdapter39 -start -n NNadk39

Table 4-13: Details for installing, starting, stopping, and testing service

	Case Description	Example	Expected Behavior
1	Install a service with no description	NNSYAdapter39 -install	Should install a service with no parameters service name = NNSYADK serviceDisplayName = NNSY-ADK-Service
2	Install a service with config file parameter	NNSYAdapter39 -install -p test.dat	Should install a service with test.dat parameter service name = NNSYADK serviceDisplayName = NNSY-ADK-Service
3	Install a service with config file parameter and serviceName	NNSYAdapter39 -install -n testSvc -p test.dat	Should install a service with test.dat parameter service name = testSvc serviceDisplayName = NNSY-ADK-Service
4	Install a service with config file parameter and serviceDisplayName	NNSYAdapter39 -install -d testSvcDisplay -p test.dat	Should install a service with test.dat parameter service name = NNSYADK serviceDisplayName = testSvcDisplay

	Case Description	Example	Expected Behavior
5	Install a service with config file parameter, serviceName, and serviceDisplayName	NNSYAdapter39 -install -n testSvc -d testSvcDisplay -p test.dat	Should install a service with test.dat parameter service name = testSvc serviceDisplayName = testSvcDisplay
6	Start a service without installing	NNSYAdapter39 -start -n testSvc	Error message noting that the service is already installed.
7	Start an installed service	NNSYAdapter39 -start -n testSvc	Service should start successfully.
8	Start a service that is running	NNSYAdapter39 -start -n testSvc	Error message noting that the service is already running.
9	Start a service without the service name	NNSYAdapter39 -start	If a service named NNSYADK is installed, it will start successfully. If that service is not installed, an error message that the service is not installed is generated.
10	Start a service that is installed without parameters		Service start should be successful. The .nml error log should provide the information that the config file is not specified.
11	Start a service with the transport manager not running		Service start should be unsuccessful. The .nml error log should provide the information.
12	Start a service with some parameters missing from the config file		Service start should be unsuccessful. The .nml error log should provide the information.
13	Stop the running service	NNSYAdapter39 -stop -n testSvc	Service should stop successfully.
14	Stop an already stopped service	NNSYAdapter39 -stop -n testSvc	Error message that the service is not running.
15	Find the status of a service	NNSYAdapter39 -status -n testSvc	Service status should be written to stdout.
16	Remove a stopped service	NNSYAdapter39 -remove -n testSvc	Service should be removed.
17	Remove a running service	NNSYAdapter39 -remove -n testSvc	Service should be stopped and removed.
18	Remove a service that is not installed	NNSYAdapter39 -remove -n testSvc	Error message that the service is not installed.
19	Perform tests for multi-threaded NNSYAdapter39	Use a .dat file with the following parameters set:: MultiThreaded = TRUE InboundTransportID = transport1,transport2,transport3	Should get the respective desired behaviors.

	Case Description	Example	Expected Behavior
20	Install multiple services	NNSYAdapter39 -install -name broker1 -d broker1 -p broker1.cfg NNSYAdapter39 -install -name broker2 -d broker2 -p broker2.cfg	The services broker1 and broker2 should be installed.
21	Remove the services		Services are not in Control Panel/Services list.

Deploying the Adapter

The steps for deploying your adapter will be determined by the requirements of your specific adapter. All adapters developed using the e-ADK require that the adapter runtime environment (ARE) be installed on the same machine as the adapter.

Redistributing the Adapter Runtime Environment

After you have developed your adapter, you must package both your adapter and the ARE. To separate the ARE from the e-ADK package you received on the e-ADK Current Release CD, run the script to copy the ARE to a separate directory. The new directory will contain the ARE that you will package with your adapter.

Before you begin, create a directory for the ARE.

❖ **To copy the image from the e-ADK Current Release CD sent with the e-ADK:**

1 Insert the CD into your CD-ROM drive.

2 Run the script in the CD-ROM directory. Use these scripts for the following platforms:

- For Windows:

  are_redist.cmd

- For Solaris, HP, and AIX:

  are_redist.sh

3 Verify that NNSY_are39 is created in the directory that you have set up to run your adapter.

After you have created NNSY_are39 in a directory, you can use this install image to install the ARE to other machines. This image can be burned to CD to be delivered with your adapter.

When you ship your adapter and the ARE, you either include the ARE in your installation package or provide instructions for installation of both the adapter and the ARE. See the *e-ADK Installation Guide* for additional information.

Message Acknowledgement

e-ADK 3.9 provides the ability to set acknowledgement of messages. This feature provides verification of the status of messages that are sent to transports using Acquire and Process modes. The acknowledgement is sent both when the message is successful (ack) and unsuccessful (nack).

In some cases, an adapter might need to take a specific action only if the message was successfully received by the transport. If the message fails, another action is required. For example, when an order is placed online, information needs to be sent to a purchase order transport. If the message is successfully sent to the transport, then an inventory action takes place. If the message fails on the purchase order transport, an alert action is produced. In this case, if the e-ARE sends an acknowledgement (ack), then the inventory action takes place. If the e-ARE sends a negative acknowledgement (nack), the alert action takes place.

Exception Handling and Logging

Informational messages, warnings, and errors are logged at strategic points during processing. Because the e-ADK functions have a boolean return value, errors must be reported by throwing an exception. Exceptions are thrown at critical points during processing, such as failure to get from a transport and failure to put to a transport, and are caught in the Adapter Shell.

e-ADK supports the following error handling:

- Transport retries

- Adapter retries
- User configurable

It also handles the following common OT errors:

- Full transport
- Transport disable for put and/or get
- Transport manager shutdown

Handling Exceptions

The e-ADK is designed to handle exceptions using the throw/catch paradigm. Errors must be reported by throwing an exception. The preferred way to handle errors inside the e-ADK code is to use one of the throw macros:

- NNADK_THROW_CRITICAL ("text")
- NNADK_THROW_WARNING ("text")
- NNADK_THROW_RETRY ("text")

A critical error usually relates to the target system; for example, the SAP or Siebel server is down, the database is down, or some other problem that would cause all subsequent messages to also fail.

Non-critical errors usually relate to data in the specific message containing an error such as a required field missing. With non-critical errors, the next message is likely to succeed.

The throw macros take a C-string, or character array, write the string to the log file, and either shut down the adapter or continue processing. To use these macros, NNADKMacro.h and NNADKLogging.h must be included. In any mode, the user can ignore the current message and continue processing by returning TRUE to the Adapter Shell. This runs the risk of lost data: the throw macros guarantee that the message gets placed on the failure transport. If the message cannot be delivered, the result of ignoring the message is that the data is neither sent to the target system nor sent to a transport. In that case, the data is lost.

Under certain conditions, an adapter may throw a critical exception that contains a value in the InformationCode1 field. This indicates a non-fatal error and, because of the value, the e-ADK keeps processing. The number of retries you have set is used.

NNADK_THROW_CRITICAL	Critical problems are handled using the NNADK_THROW_CRITICAL macro. NNADK_THROW_CRITICAL takes a character string, logs the string to the standard logfile, and then shuts down the shell. If a failure transport is defined for Deliver or Process mode, the message retrieved from the transport is placed on the failure transport before shutting down.
NNADK_THROW_WARNING	NNADK_THROW_WARNING takes a character string, logs the string to the standard logfile, and then continues processing. If a failure transport is defined for Deliver or Process mode, the message retrieved from the transport is placed on the failure transport before continuing.
NNADK_THROW_RETRY	NNADK_THROW_RETRY takes a character string, logs the string to the standard logfile, and then returns to the Adapter Shell. If adapter_wait_time and maximum.adapter.retries are set in the cfg file, the Adapter Shell retries calling the adapter function for a maximum number of times specified by maximum.adapter.retries waiting the number of seconds specified by adapter_wait_time before each retry. If those keys are not set, the Adapter Shell terminates.

Methods of Handling Errors

The preferred way to handle errors inside the e-ADK code is to use one of the throw macros in NNADKMacro.h. The e-ADK provides throw macros that take various arguments, write a string to the log file, and either shut down the adapter or continue, depending on the macro used. NNADKLogging.h must also be included if these macros are used. In any mode, the user can ignore the current message and continue processing by returning TRUE to the Adapter Shell.

This table provides more detailed information about the macros that can be used:

Table 4-14: e-ADK 3.9 Macros

Macro	Argument	Logged Info	Effect on Current Message	Effect on Processing
NNADK_THROW_CRITICAL	String	String	Sent to the failure transport if one is defined	e-ADK shuts down
NNADK_THROW_CRITICAL_USERMESSAGE	MsgSet MsgIndex	Using the MsgIndex, a string is retrieved from the indicated catalog file and logged	Sent to the failure transport if one is defined	e-ADK shuts down
NNADK_THROW_CRITICAL_USERMESSAGE_PARAMS	MsgSet MsgIndex Params	Using the MsgIndex, a string is retrieved from the indicated catalog file, merged with the Params, and logged.	Sent to the failure transport if one is defined	e-ADK shuts down
NNADK_THROW_WARNING	String	String	Sent to the failure transport if one is defined	Processing in the e-ADK continues
NNADK_THROW_WARNING_USERMESSAGE	MsgSet MsgIndex	Using the MsgIndex, a string is retrieved from the indicated catalog file and logged	Sent to the failure transport if one is defined	Processing in the e-ADK continues
NNADK_THROW_WARNING_USERMESSAGE_PARAM	MsgSet MsgIndex	Using the MsgIndex, a string is retrieved from the indicated catalog file, merged with the Params, and logged	Sent to the failure transport if one is defined	Processing in the e-ADK continues
NNADK_THROW_RETRY	String	String	Message is retried	Processing in the e-ADK continues
NNADK_THROW_RETRY_USERMESSAGE	MsgSet MsgIndex	Using the MsgIndex, a string is retrieved from the indicated catalog file and logged	Message is retried	Processing in the e-ADK continues

Macro	Argument	Logged Info	Effect on Current Message	Effect on Processing
NNADK_THROW_RETRY_USERMESSAGE_PARAMS	MsgSet MsgIndex Params	Using the MsgIndex, a string is retrieved from the indicated catalog file, merged with the params, and logged	Message is retried	Processing in the e-ADK continues

Logging

Runtime log messages are stored in a catalogue file. NNSYMessageLog.nml is the New Era of Networks standard log file. This file appears in the local directory when an adapter is run. The file resides in a directory specified by the following:

<"NNSY_ROOT">/NNSYCatalogues/<language code>, where

<"NNSY_ROOT">/NNSYCatalogues/en_US is an example.

If the NNSY_ROOT environment variable is not set, the e-ADK attempts to find the catalogue files in the runtime directory ./NNSYCatalogues/xxxx, where xxxx is the code set (for example: en_US). If the files are not found, the following message is put to the output log file, NNSYMessageLog.nml:

"Could not find a valid message catalogue containing messages for component ’ADK Message Set’."

If NNSYMessageLog.nml file cannot be created or cannot be written to, log information is written to standard error. Developers can create a NNSYMessageLog. nml file that is write-protected, and the output will be channeled to the screen.

New Era of Networks has a standard format for logged messages. The format is:

CCYYMMDDhhmmsslllZ|<process name>|<process ID>|<thread ID>|

<correlation ID>|<class>|<code>|<severity>|<file>|<line>|

<text>

Each logged message is contained on a single line that can wrap. Each message is delimited by a newline character.

Using Tools for Debugging

Before using the debug functionality, which can be used only on Windows NT, verify that the debug libraries (nnsy\adk39\debug) are set up to run instead of the release libraries (nnsy\adk39\bin). The default location is c:\nnsy\adk39\debug. You must have both the bin and debug directories set up to run debug.

Mixing debug-version libraries with release-version libraries causes an assertion exception in the file dbgheap.c. If running in release mode, use all release libraries and executables. If running in debug mode, use all debug libraries and executables.

The following tools are available for debugging:

- -trace Option
- Using File Driver for Debugging

-trace Option

Use the -trace option on the command line to get extra information in the NNSYMessageLog.nml file.

The following sample command line shows the use of the -trace option:

NNSYAdapter39 -file=schema.dat -trace

NNADK_TRACE Macro

The NNADK_TRACE macro enables you to insert statements into the nml file. NNSYAdapter39 must be run with the -trace option for these statements to appear.

NNADK_TRACE is based on the vprintf() function. It takes a format string and a variable list of arguments and outputs a formatted string to the nml file. The following is an example:

```
STL_STRING   InFileName=config.getStringValue("Adapter.in_file_name");

NNADK_TRACE("Infile--%s\n", InFileName.c_str());
```

Assuming that the value of the Adapter.in_file_name key is "s.txt", the e-ADK would place the following string in the nml file:

"Infile--x.txt"

See a C or C++ reference for more information about vprintf().

Using File Driver for Debugging

File drivers can be used for testing during Acquire, Process, and Deliver modes. Use the File Driver for your Transport to output your data to a file instead of to a transport.

You set up the file driver to allow this function. Sample configuration files that illustrate how to set up a file driver are provided in the examples directory as acquirefile.dat, processfile.dat, and deliverfile.dat.

Warning! If test.drive=true in Acquire, Process, or Deliver mode configuration files, you will receive the critical error:

Test Drive unavailable. Set test.drive=false. Use the file driver instead.

Reviewing the Schema Tree

This option is available only in Schema mode. You can review a schema tree before it is loaded into the repository using one of the following methods:

- To display your schema information on the screen but not load the schema tree into the repository, add the following under Adapter in the configuration file:

  test.drive=true

- Use the DTD Schema Loader to write out a DTD file that represents the schema tree created by a schema. Change the following information in the configuration file:

  SchemaLoader.Library=adk39dtdsl

  SchemaLoader.Factory=DTDSchemaLoader_Factory

- Use the XML Schema Loader to write out an xml file that represents the schema tree that would be loaded into the repository. Change the following information in the configuration file:

  SchemaLoader.Library=adk39xmlsl

  SchemaLoader.Factory=XMLSchemaLoader_Factory

- Use the NCM Schema Loader to write out an xml-like file that represents the schema tree that would be loaded into the repository. Change the following information in the configuration file:

  SchemaLoader.Library=adk39ncmsl

  SchemaLoader.Factory=NCMSchemaLoader_Factory

CHAPTER 5

Troubleshooting

Use the information in this chapter to attempt to resolve problems before calling Technical Support. The following troubleshooting information is available in this chapter:

Using Error Messages

If you have difficulty with e-ADK, begin by reviewing error messages.

1 Check the NNSYMessageLog.nml file for error messages.

2 Use Trace so that more detailed messages are written to the log for easier troubleshooting.

To use Trace, at the command line, add *-trace*.

Verifying the Environment

Verify the following:

1 Verify that the adapter is correctly installed and configured for your environment.

2 Check the environment variables.

The environment variables and their paths are listed at the end of the installation instructions for each platform.

3 Verify paths.

Be sure you have used the same directories as those suggested in the installation guide.

4 Verify the configuration file information.

- The adapter points to your plug-in.
- Mode and data representation are correct.
- Session information is accurate.

5 Verify that the error catalogue files exist.

- The error catalogue files must be in the following location:

NNSY_ROOT/NNSYCatalogues/en_US

Note The Adapter Shell does not display errors on the screen.

6 Verify your data source files and data target file.

Searching for Versions

You can use nnversion to find out which version of a specific file you are using. This is especially helpful if you have a number of versions on your machine and need to provide technical support with the version you are using. The nnversion.pl replaces NNIdent and NNWhich.

To run nnversion.pl, type the following:

```
perl nnversion.pl libndo 10.so
```

Order and Format of Fields

When you input a message for Deliver or Process mode, the order of the fields is important and must be exactly the same order as was loaded into the formatter.

Use *ncm to verify that the formats match. If you are using e-Biz Integrator, you can set the "Random Field Order" in the formatter.

Schema Does Not Exist Error

The following error may appear when you are using the NCF Input Serializer in Deliver or Process mode:

```
Schema Does Not Exist in Repository(ADK)
```

The following information message appears a few lines above the message:

```
message type = ADK
```

Possible Issues

The NCF Input Serializer must have Schema information defined in a schema repository. The message type is a concatenation of <prefix> + <'.IC'> + <schema name> used when loading the schema. This schema is found based on the message type defined for the message. The schema search is based on the prefix name as well as the message type name. The following search order is used:

- If the message coming from the transports has the OPT_MSG_TYPE property (option) defined in the RFH for MQSeries message, that message type will be used as the schema name.
- If the message does not contain the OPT_MSG_TYPE property, the msg.type key in the configuration file is used.
- If the key is not preset, "ADK" is the default message type.

Possible Resolutions

Correct the problem with one of the following:

- Add the OPT_MSG_TYPE on the incoming message (make sure MQS_PROPAGATE = PROPAGATE in the putqueue action if running MQSI). Set the prefix key in the configuration file.
- If no OPT_MSG_TYPE is set, set the msg.type key and the prefix key in the configuration file.
- Verify that the .nmc file has the correct name and is located in the correct directory (<NNSY_ROOT>\NNSYContentRepository\<AppGrp>) on the correct machine.

Using -trace Option

Use the -trace option on the command line to get extra information in the NNSYMessageLog.nml file on the console. The following sample command line shows the use of the -trace option:

```
NNSYAdapter39 -file=schema.dat -trace
```

Reviewing the Schema Tree

This option is available only in Schema mode. You can review a schema tree before it is loaded into the repository using one of the following methods:

1 To display your schema information on the screen but not load the schema tree into the repository, add the following under Adapter in the configuration file:

```
test.drive=true
```

2 Use the DTD Schema Loader to write out a DTD file that represents the schema tree created by a schema. Change the following information in the configuration file:

```
SchemaLoader.Library=adk39dtdsl
SchemaLoader.Factory=DTDSchemaLoader_Factory
```

3 Use the XML Schema Loader to write out an xml file that represents the schema tree that would be loaded into the repository. Change the following information in the configuration file:

```
SchemaLoader.Library=adk39xmlsl
SchemaLoader.Factory=XMLSchemaLoader_Factory
```

4 Use the NCM Schema Loader to write out an xml-like file that represents the schema tree that would be loaded into the repository. Change the following information in the configuration file:

```
SchemaLoader.Library=adk39ncmsl
SchemaLoader.Factory=NCMSchemaLoader_Factory
```

Using File Driver for Debugging

File drivers can be used for testing during Acquire, Process, and Deliver modes. Use the File Driver for your Transport to output your data to a file instead of to a transport.

Set up the file driver to allow this function. Sample configuration files that illustrate how to set up a file driver are provided in the examples directory as acquirefile.dat, processfile.dat, and deliverfile.dat.

Warning! If test.drive=true in Acquire, Process, or Deliver mode configuration files, you will receive the critical error:

```
Test Drive unavailable
```

Set test.drive=false and use the file driver instead.

Glossary

accelerator keys — A combination of keystrokes that bypasses the menu system to carry out an action.

action (New Era of Networks Adapter for Databases) — An action is a SQL statement defined as part of a format within a format specification file.

action (New Era of Networks Rules) — An action defines a function to perform in a subscription. An action must contain at least one option name-value pair.

adapter — A component that provides an interface between an internal application and external applications or messaging systems. An adapter detects events and validates event contents. In Sybase Enterprise Event Broker, adapters pass events to an inflow processor. Adapters also receive events at outflow processors and export the events to external applications.

adapter runtime environment (ARE) — The adapter runtime environment includes the Adapter Shell, e-ADK Shared Libraries, shared data dictionary, and error catalogue files. The ARE is required for testing and running the adapters that are run on the customer's server.

alternative format — A special form of compound format where one format in a set of alternatives applies to a message. For example, if the alternative format is named A, it may contain component formats B, C, and D. A message of format A may actually be of variation of only B, C, or D.

American Standard Code for Information Interchange (ASCII) — The standard code used for information interchange among data processing systems, data communication systems, and associated equipment. The code uses a coded character set consisting of 7-bit coded characters (8 bits including a parity check). The ASCII set consists of control characters and graphic characters.

ANSI ASC X12 "997" Functional Acknowledgement. An EDI functional acknowledgement that indicates successful transmission of a file and whether or not the file passed syntactical edits within the receiver's translator. Similar in function to the green return receipt one receives from the U.S. Postal Service when sending certified mail. Getting the card back indicates the letter was delivered – it does not validate that the contents of the letter were read, understood, or any action was taken as a result of receiving the letter. There are three types of functional acknowledgements. The first type is Group Accepted. The second type is Group Accepted, but Errors Were Noted. The third type is Group Rejected.

applet A Java program that runs within the web browser. When using Java on the Web, an applet is an HTML-based program built with Java that a browser temporarily downloads to and runs fro a user's hard disk. Java applets can be used to add background music, real-time video displays, animation, and interactivity such as calculators and games to Web pages without having to send a user request back to the server.

application A packaged application, database, protocol, file, or other data source.

application endpoint Also called endpoint. These are applications such as General Ledger, Patient Admitting, or Materials Planning that run on computers attached to a network. Endpoints can be connected to other endpoints using adapters and an integration engine.

application group A logical grouping of applications used to organize rules.

application link enabling (ALE) The communication layer for inbound and outbound data to and from an SAP system.

application program interface (API) The interface (calling conventions) by which an application program accesses services. An API is defined at source-code level and provides a level of abstraction between the application and the kernel or other privileged utilities to ensure portability of the code.

argument In New Era of Networks Rules, an argument is evaluation criteria made up of fields from a message and associated operators. It is a standard encoding for alphanumeric data.

asynchronous In electronic messaging, a method of operation in which receiving applications are loosely coupled and independent. The receiver need not respond immediately to a message, and the sender does not have to wait for a response before proceeding with the next operation. Compare to synchronous.

batch processing A method of handling computer operations in which requests for operations are grouped for periodic processing. Compare to transaction processing.

bean

A reusable software component. Beans can be combined to create an application.

binary large object block (BLOB)

A collection of bytes containing data. It is generally stored as a flat file containing multiple, delimited records and is terminated by a number that specifies the length of the file.

binding

The association of a client and a server.

bridge

A physical link between network servers that are not tightly connected via TCP/IP.

A network description parameter that describes the physical link.
A logical representation of a connection between two network servers that reside on separate network LANs.

A software component that allows one integration engine to communicate with another.

broker

See integration server.

A type of middleware that connects clients and servers.

A program that executes in the background. Also known as agents, services, or daemons, brokers only activate when defined system conditions become true. For example, a broker may activate when the system clock reaches 2:00 a.m. every other Saturday. Another broker may activate when the system senses the arrival of e-mail

business application programming interface (BAPI)

An open, business-object interface used to access SAP R/3 business processes and data from external systems.

business object

An application-level component you can use in unpredictable combinations. A business object is independent of any single application. Business objects provide a natural way for describing application-independent concepts such as customer, order, competition, money, payment, and patient. They encourage a view of software that transcends tools, applications, databases, and other system concepts.

business to business (B2B)

Companies using the Web to deliver products, services, support, and information over the internet to other companies.

business to business integration (B2Bi)

The integration of applications, including data and process integration, between enterprises.

character set

A set of specific (usually standardized) characters with an encoding scheme that uniquely defines each character. ASCII is a common character set.

class

In object-oriented programming, a category of objects. For example, there might be a class called shape that contains objects which are circles, rectangles, and triangles. The class defines all the common properties of the different objects that belong to it.

client

A system or process that requests a service from another system or process.

client application

Software that is responsible for the user interface, including menus, data entry screens, and report formats. It also is an application that sends requests to another application that acts as a server. See also

client-server

A paradigm for distributed computing under which the system is split between one or more server tasks, which accept requests according to some protocol, and client tasks, which request information or actions. Clients and servers can be placed independently on network nodes.

A network architecture in which one or more computers (servers) accept requests for services from one or more workstations (clients). This may also refer to a back-end application (server) that accepts requests for information from a front-end application (client).

commit

An instruction to a database to make permanent all changes made to one or more database files since the last commit or rollback operation and to make the changed records available to other users. compare with rollback.

common gateway interface (CGI)

A standard for running external programs from a World Wide Web HTTP server. CGI specifies how to pass arguments to the executing program as part of an HTTP request. It also defines a set of environment variables. Commonly, a program will generate some HTML which will be passed back to the browser but it can also request URL redirection.

CGI allows the returned HTML (or other document type) to depend in any arbitrary way on the request. The CGI program can, for example, access information in a database and format the results as HTML. A CGI program can be any program which can accept command line arguments.

common object request broker architecture (CORBA)

CORBA is a distributed-objects standard developed and defined by the Object Management Group (OMG). CORBA provides the mechanism by which objects transparently make request and receive responses, as defined by OMG's Object Request Broker (ORB). The CORBA ORB is an application framework in which objects can communicate with each other, even if they are written in different programming languages, are running on different platforms, reside at different locations. or were developed by different vendors.

common programming interface for communictions (CPI-C)

An SNAplus API which allows peer-to-peer communication between TPs in a network. CPI-C uses the same underlying communications elements (modes, LUs and security information) as APPC, and the information transferred between CPI-C applications is in the same format as information transferred between APPC TPs. This means that a CPI-C application may communicate with an APPC application, and neither application needs to know which API the other is using.

component

In programming and engineering disciplines, a component is an identifiable part of a larger program or construction. Usually, a component provides a particular function or group of related functions.

In object-oriented programming and distributed object technology, a component is a reusable program building block that can be combined with other components in the same or other computers in a distributed network to form an application. Examples of a component include a single button in a graphical user interface, a small interest calculator, an interface to a database manager.

Components can be deployed on different servers in a network and communicate with each other for needed services. A component runs in a context called a container. Examples of containers include pages on a Web site, Web browsers, and word processors.

component object model (COM)

An architecture for defining interfaces and interaction among objects implemented by widely varying software applications. A COM object instantiates one or more interfaces, each of which exposes zero or more properties and zero or more methods. All COM interfaces are derived from the base class IUnknown. MMC is built on the COM foundation.

configure

To define to a system the devices, optional features, and programs installed on the system.

configuration file

A file that specifies the characteristics of a system or subsystem.

configuration set

A section into which service library configuration files are divided.

connection string

A connection string is a string version of the initialization properties needed to connect to a data source and enables you to easily store connection information within your application or pass it between applications. Without a connection string, you would be required to store or pass a complex array of structures to access data. The basic format of a connection string is based on the ODBC connection string. The string contains a series of keyword/value pairs separated by semicolons. The equals sign (=) separates each keyword and its value.

connectivity

The capability to attach a variety of functional units without modifying them.

console A computer terminal used to monitor and control a computer or network.

cross-platform Used to describe programs that can execute in dissimilar computing environments.

daemon A daemon is a program that is always running in the background on the system, initialized, and waiting for something to do.

database The file or physical allocation of space on a disk intended to hold schema.

database management system (DBMS) A computer-based system for defining, creating, manipulating, controlling, managing, and using databases. It is a program that lets one or more computer users create and access data in a database. The DBMS manages users requests (and requests from other programs) so that users and other programs are free from having to understand where the data is physically located on storage media and, in a multi-user system, who else may also be accessing the data. In handling user requests, the DBMS ensures the integrity of the data (that is, making sure it continues to be accessible and is consistently organized as intended) and security (making sure only those with access privileges can access the data). The software for using a database can be part of the database management system or it can be a stand-alone database system. Contrast with relational database management system.

data dictionary A collection of descriptions of the data object or items in a data model for the benefit of programmers and others who might need to refer to them. A data dictionary can be consulted to understand where a data item fits in the structure and what values it may contain.

data replication The process of copying data to remote locations. The copied (replicated) data is then kept synchronized with the primary data. Data replication is distinct from data distribution. Replicated data is stored copies of data in particular sites throughout a system and is not necessarily distributed data. See also data distribution and transaction replication.

data server A database management system program that responds to client requests. See also local area network.

data source name (DSN)

A textual string that is used to reference the data source by application programs. A unique identifier must be provided for each data source. A data source consists of the data a user wants to access, its associated database management system (DBMS), the platform on which the DBMS resides, and the network (if any) used to access that platform. Each data source requires that a driver provide certain information in order to connect to it. At the core level, this is defined to be the name of the data source, a user ID, and a password. ODBC extensions allow drivers to specify additional information, such as a network address or additional passwords.

data type

A keyword that identifies the characteristics of stored information on a computer. Some common data types are char, int, smallint, date, time, numeric, and float. Different databases support different datatypes.

decrypt

Decryption is the process of converting encrypted data back into its original form, so it can be understood

delimiter

One or more characters marking either the end or beginning of a piece of data.

deserialize

Process of walking a tree (for example, the DataTree of the NNDOObject) and writing the data for each node to a string buffer. This generates a buffer containing the wire format representation of the data in the tree.

distributed COM (DCOM)

A protocol that enables software components to communicate directly over a network in a reliable, secure, and efficient manner. Based on the Open Software Foundation's DCE-RPC specification, DCOM deploys across heterogeneous platforms and works with both Java applets and ActiveX components.

distributed function call (DFC)

A distributed function call has three major components: the DFC command definition in a client module that identifies the DFC command's name and its arguments; the DFC service in a server module that receives the command and its arguments and processes them in some way; and the Impact Manager Configurator entries that identify where the DFC command is serviced.

document type definition (DTD)

A document type definition is a specific definition that follows the rules of the Standard Generalized Markup Language (SGML). A DTD accompanies a document and identifies what the codes (or markup) are that separate paragraphs and identify topic headings and how each is to be processed.
A generated document that specifies the grammatical structure of other SML documents. A hypertext markup language entity to describe the document type.

domain

A group of computers and other devices that are networked and managed as a unit, with policies and rules specific to the unit.

domain name system or service (DNS)
An internet service that translates domain names into IP addresses.

driver
A program that interacts with a particular device or specially (frequently optional) kind of software. The driver contains the special knowledge of the device or special software interface that programs using the driver do not. In a personal computer, a driver is often packaged as a dynamic link library (DLL) file.

dynamic-link library (DLL)
A module containing functions and data that can be loaded at run time by a calling module (an executable file or another dll).
A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a dynamic link library can be shared by several applications simultaneously.

EAS message service
A messaging system native to the EAServer.

EAServer
This is the Sybase web application server. An integrated set of application servers used to deploy web applications.

EDI document
An ANSI ASC X12 transaction.

EDIFACT
The United Nations' counterpart to ANSI's X12. Another format for businesses exchanging data electronically.

e-FTP
Component of the EC Gateway for UNIX which facilitates FTP client operations.

electronic data interchange (EDI)
A standard for exchanging business data. EDI is the process of sending and receiving electronic messages in standardized formats between business partners. Electronic messages are exchanged between companies' computer systems and are used to replace traditional paper based transactions and sometimes telephone call confirmations of transactions. EDI solutions have allowed businesses to exchange information more accurately and in a matter of minutes or hours rather than days

element
Basic building block of an X12 transaction. Each segment is comprised of one or more elements. Its segment and then its position within that segment describe each element. A transaction-specified delimiter separates these elements.

element separator
The character that will appear at the end of each element within an X12 transaction. For inbound X12 documents, this character definition is read directly from the envelope. For outbound documents, customer-specific element separators are retrieved from the trading partner database.

encrypt Encryption is the conversion of data into a form that cannot be easily intercepted by unauthorized people.

endpoint An application that can be directly accessed or updated by an acquisition or delivery adapter (via an integration server).

enterprise application integration (EAI) EAI involves the integration of applications (including data and process integration) within an enterprise.

enterprise information system The systems that provide the information infrastructure for an enterprise. Enterprise resource planning systems, relational database management systems, and legacy information systems are examples of enterprise information systems.

Enterprise Java Beans Specification for creating server-side scalable, transactional, multi-user, secure enterprise-level applications. Provides consistent component architecture framework for creating distributed n-tier middleware. Low-level details are separated from business logic.

envelope The header and trailer information contained within an ANSI ASC X12 transaction. The envelope is critical for routing, trade agreement determination, and message management.

event An entity that is sent into the system and drives the business processing. It consists of a name, scope, and attributes.

event definition language (EDL) The format used by New Era of Networks Process Server to describe a business process.

event set A grouping of events that can be shared by more than one business process.

environment variable A variable that describes how an operating system runs and the devices it recognizes.

extended binary-coded decimal interchange code (EBCDIC) An IBM code for representing characters with numerical values. This code is used mainly on IBM computers.

eXtensible markup language A simplified subset of Standard Generalized Markup Language (SGML) that provides a file format for representing data, a method for describing data structure, and as a mechanism for extending and annotating HTML with semantic information.

	As a universal data format, XML provides a standard for the server-to-server transfer of different types of structured data so that the information can be decoded, manipulated, and displayed consistently and correctly. In addition, it enables the development of three-tier Web applications, acting as the data transfer format between the middle-tier Web server and the client.abend.
field	The smallest possible container for information. You can use a field in more than one table. If the first_name field exists in one table, for example, you can use the same field in other tables.
file transfer protocol (FTP)	A TCP/IP utility that moves files efficiently between machines.
flat file	The file produced by basic EDI translation software to serve as input to the interface. Usually has the same fields as the standard but has each field expanded to its maximum length. A computer file where all the information is run together in a single character string.
flat format	A format containing only fields and associated controls. Flat input formats are composed of fields with associated control input controls. Flat output formats are composed of fields with associated output controls.
format	Formats describe how messages are constructed. Input formats describe how to separate input messages into their component parts. Output formats describe how to build output messages from the parsed components of the input message..
functional acknowledgement	See ANSI ASC X12 “997”.
function module	An ABAP function accessible using RFC. The interface to a specific function module is defined in the R/3 data dictionary.
function module group	A collection of related business functions.
function wrapper	An option available with DFC command definition. It automatically builds a callback function that, upon receiving an altered data condition (CBE_ALT), issues the DFC command and checks for an error condition return. The function name is the same as the DFC command, but with a preceding underscore character (_dfc_cmd_name). The user can attach the function directly to a callback function property for any control in a user interface's form object.
get	A request for the next message in a queue. Compare to put.

globalization The combination of internationalization and localization. See also internationalization and localization.

graphical user interface A type of computer interface consisting of a visual metaphor of a real-world scene, often of a desktop. Within that scene are icons, representing actual objects, that the user can access and manipulate with a pointing device.

heterogeneous Composed of different parts of different kinds. Having dissimilar constituents.

Health Level 7 The standard message format used by many healthcare applications.

host A host computer is directly connected to the Internet that provides services to other computers on the network, such as e-mail connections or access to program and data files. Each host computer has a unique Internet address, or IP, and a unique domain name, which identifies the computer to other computers and users on the Internet. Host means any computer that has full two-way access to other computers on the Internet.

Hypertext Markup Language (HTML) The language used to mark a document so it can be published on the World Wide Web (WWW) and viewed with a browser.

HyperText Transport (or Transfer) Protocol HyperText Transport (or Transfer) Protocol is the set of rules that governs the exchange of text, graphic, sound, and video files on the World Wide Web.

interactive development environment (IDE) A User Interface that allows users to build complex projects or programs by filling out forms in a Windows program. Examples are the MSG-IDE tool (for creating custom adapters) and the TRAN-IDE tool (for creating production objects).

idempotent This term means unchanged when multiplied by itself. In the context of New Era of Networks products, the term idempotent refers to the idempotent attribute, which specifies that an operation can be safely executed any number of times. If an operation is idempotent, the server does not need to save results and the client does not need to issue acknowledgements (improving performance). An example of an idempotent operation is one that simply reads a value. An operation that increments a value, for example, is not idempotent.

identifier An identifier data element always contains a value from a predefined list of values (codes) that are maintained by the ASC X12 or some other body recognized by the X12 Committee.

ido The extension given to IDoc metadata files.

instance An Oracle-specific term for a set of memory structures and background processes that access a set of database files. Compare to database sever.

integrated development environment system (IDES)

A programming environment integrated into an application. For example, Microsoft Office applications support various versions of the BASIC programming language. You can develop a WordBasic application while running Microsoft Word. .

integration point

An integration point is an entry point into a computer system. It typically consists of the login information required to establish a connection with a software system so that information can be transferred into and out of the system.

interactive development environment (IDE)

A User Interface that allows users to build complex projects or programs by filling out forms in a Windows program. Examples are the MSG-IDE tool (for creating custom adapters) and the TRAN-IDE tool (for creating production objects).

Intermediate Document (IDoc)

A data container used by R/3 applications used by ALE to send and receive information. MQSeries Link for R/3 works with IDocs only.

International Organization for Standardization (ISO)

An organization of national standards bodies from various countries established to facilitate international exchange of goods and services and develop cooperation in intellectual, scientific, technological, and economic activity.

internationalization

The process of extracting locale-specific components from the source code and moving them into one or more separate modules, making the code culturally neutral so it can be localized for a specific culture. See also globalization. Compare with localization.

IP address

A 32-bit number that identifies each sender or receiver of information that is sent in packets across the Internet. An IP address has two parts: the identifier of a particular network on the Internet and an identifier of the particular device, which can be a server or a workstation, within that network.

Internet server application programming interface (ISAPI)

Microsoft's programming interface between applications and their Internet Server. Active Servers created with ISAPI extensions can be complete in-process applications themselves, or can "connect" to other services. ISAPI is used for the same sort of functions as CGI but uses Microsoft Windows dynamic link libraries (DLL) for greater efficiency. The server loads the DLL the first time a request is received and the DLL then stays in memory, ready to service other requests until the server decides it is no longer needed. This minimizes the overhead associated with executing such applications many times.

item

For users: Use this term when referring to specific content in the console tree. Do not use the terms node or namespace. If possible, refer to the actual name of the item in the tree unless you must use an explicit term. To direct users to an item, you should write out the entire path to the item.

Java

Developed by Sun Microsystems, Java is an object-oriented programming language, similar to C++. Java-based applications, or applets, can be quickly downloaded from a Web site and run using a Java-compatible Web browser such as Microsoft Internet Explorer or Netscape Navigator. Java applets are the most widespread use of Java on the Web. Java programs or source code files (.java) are compiled into a format known as bytecode files (.class). These files, once complied, can be executed by a Java interpreter. Most operating systems, including Windows, Macintosh OS, and UNIX, have Java interpreters and run-time environments known as Java Virtual Machines.

Java2 Enterprise Edition

Sun Microsystems, Inc. Java™ 2 Platform, Enterprise Edition (J2EE) defines the standard for developing multi-tier enterprise applications. J2EE simplifies enterprise applications by basing them on standardized, modular components, by providing a complete set of services to those components, and by handling many details of application behavior automatically, without complex programming.

Java ARchive (JAR)

A file containing a collection of Java class library files.

Java Connector Architecture

Sun Microsystems, Inc. architecture that defines a uniform way to integrate J2EE application servers with enterprise information systems. A compliant EIS can plug into any application server that supports the connector architecture. An application server that conforms to this standard can connect to any EIS that provides a standard resource adapter.

Java Development Kit

Sun Microsystems, Inc.'s foundation for building and deploying client-side enterprise applications with cross-platform compatibility.

Java Dynamic Management™ Kit

The Java Dynamic Management™ Kit is the foundation for building and distributing network management intelligence into applications, networks, and devices.

Java Naming and Directory Interface

One of Sun Microsystems, Inc.'s application program interfaces (APIs): Java Database Connectivity (JDBC™) Java Remote Method Invocation (RMI) Java Interface Definition Language (JavaIDL) Java Naming and Directory Interface (JNDI).

JavaServer Pages™

Sun Microsystems, Inc. technology used to embed bits of Java™ code (or scriptlets) in HTML documents.

key A field that contains unique information.

library A named disk area that can contain programs and related information. A library consists of different sections, called library members.

literal One or more symbols or letters in data that represents itself.

local area network (LAN) A logical grouping of network servers. In the network, every network server must be on a LAN. Servers on the same LAN must be connected with TCP/IP.

local queue When using MQSeries Remote Queueing, a local queue is defined on the receiver side that can be referred to from the sending queue. The receiving application talks to the local queue.

localization The process of preparing an extracted module for a target environment, in which messages are displayed and logged in the user's language; numbers, money, dates, and time are represented using the user's cultural convention; and documents are displayed in the user's language. See also globalization. Compare with internationalization.

logical system A system in which applications run integrated on a common database. A client corresponds to a logical system.

logical unit A type of unit that enables end-users or programs to gain access to network resources and communicate with each other.

lookup A search done by the computer within a predefined table of values or within a data file.

LU 6.2 A protocol used for APPC and CPI-C communications between transaction programs. LU 6.2 uses the generalized data stream (GDS) format. SNAplusAPI APPC and CPI-C support LU type 6.2

mapping Specification that indicates the value for an element or attribute in a target document that is produced by a Mapper transformation. A mapping can be an XPath to a source schema item, a string literal, or a function that operates on XPaths, string literals, or functions.

mapping (EDI) The translation process from a standard format to another format. The software component that governs the conversion of application data to and from EDI interchanges is called an EDI translator. Most EDI translators provide two services: data mapping and standards formatting.

To access the data, most translators support a file interface. For outbound transactions an application writes the transaction data to a sequential text file (also called a flat-file). The translator formats the data according to the appropriate EDI syntax rules and produces an EDI file, which is ready to be communicated to a trading partner. For inbound transactions the translator verifies that the standard version and release are supported, and that the syntax of the interchange is in compliance with the standards. The translator produces a flat-file for the application as output.

To convert flat-file data to and from EDI data, a translator must understand the format of the flat-file data. This understanding is achieved in one of two ways. First, the translator might require the user to generate the flat-file according to a format defined by the translator. This means that the user must modify the application data so the translator can process it. Second, the translator might provide a tool that allows the user to specify the format of the flat-file. This tool is called a data mapper. Data mapping reduces or eliminates the programming required to integrate the translator with a business application.

message type A message type defines the layout of a string of data. The message type name in the Rules GUI is the same as the input format name in Formatter.

message A string of bytes that has meaning to the applications that use it. Messages are used for transferring information from one application to another between components in a single application. The applications can be running on the same platform or on different platforms.

message-driven bean An enterprise bean that enables asynchronous consumption of messages.

message-driven components Components that enable asynchronous, event-based processing in the application server.

message queueing A form of communication between programs. Application data is combined with a header (information about the data) to form a message. Messages are stored in queues, which can be buffered or persistent (see buffered queue and persistent queue). It is an asynchronous communications style and provides a loosely coupled exchange across multiple operating systems.

message type A message type defines the layout of a string of data. The message type name in the Rules GUI is the same as the input format name in Formatter.

messaging Software that can enable the capture and delivery of information between applications.

metadata Data that describes other data. Any file or database that holds information about another database's structure, attributes, processing, or changes.

middleware Software that facilitates the communication between two applications. It provides an API through which applications invoke services and it controls the transmission of the data exchange over the network. There are three basic types: communications middleware, database middleware, and systems middleware.

mode A method used by adapters to invoke specific methods of operation. New Era of Networks version 3.9 adapters use Acquire, Deliver, Schema, Schema Remove, and Process modes. The user guide for each adapter contains detailed information about modes and data representation.

mutex Mutex stands for mutually exclusive. This means that a computer resource can be made available to one user at a time. It can best be explained by this simple example. Person A wants to run a process script and does not want the file to be accessed by anyone else during the run time of the script. The use, therefore, locks the file. Once the process script has completed running, the computer resource can be unlocked and then becomes available for other to access.

NAK The mnemonic for ASCII character 21.

Sometimes used as the response to receipt of a corrupted packet of information.

Any message transmitted to indicate that some data has been received incorrectly, for example it may have a checksum or message length error. A NAK message allows the sender to distinguish a message which has been received in a corrupted state from one which is not received at all

An alternative is to use only ACK messages, in which case the non-receipt of an ACK after a certain time is counted as a NAK but gives no information about the integrity of the communications channel.

nesting level Level within the hierarchy of a specific component. A repeating child format and the fields within it may have a nesting level one greater than that of the parent format and any nonrepeating components of the parent format. The nesting level of the root format is one.

Network Interface Definition Language (NIDL) A binary file (NIDL.exe) used to compile ".idl type" files. IDL files are created to allow C-based programs to contain DFC functions.

New Era of Networks canonical form New Era of Networks canonical form (NCF) is a layout specification that states how data is transported to the wire.

New Era of Networks data object New Era of Networks data object (NDO) is the in-memory form of New Era of Networks canonical form.

Object Definition Language (ODL)
The New Era of Networks-provided language that users can use to build Initialization, De-initialization, Validation, Callback, and Custom functions in programs.

Object Request Broker
Software that allows objects to dynamically discover each other and interact across machines, operating systems, and networks.

Open Database Connectivity (ODBC)
ODBC is a Windows standard API that is used for SQL communication to connect applications to a variety of data sources. By using ODBC statements in a program, you can access files in a number of different databases, including Access, dBase, Excel, and Text. ODBC is based on and closely aligned with the Open Group standard Structured Query Language (SQL) Call-Level Interface. ODBC handles the SQL request and converts it into a request the individual database system understands. An open system (as opposed to a proprietary system) is one that adheres to a publicly known and sometimes standard set of interfaces so that anyone using it can also use any other system that adheres to the standard. Access is generally provided through the Control Panel, where data source names (DSNs) can be assigned to use specific ODBC drivers.

open transport
Open Transport configuration provides a means to adapt New Era of Networks applications to supported transport and transaction manager environments. This configuration capability increases the stability of the entire application by providing a single code base that maintains flexibility required by clients in a heterogeneous enterprise setting

option
An option consists of a name-value pair of data related to an action. An option name can be predefined (for Reformat and Put Message) or user-defined.

parameter
A variable that is given a constant value for a specified application and that can denote the application. Compare with property.

parent/child
Compound formats contain other flat and compound formats. If you have a compound format (X) that contains a repeating format (Y), X is the parent ot child Y.

parse
To analyze a message by breaking it down into its component fields.

permission
The level of access to an object, resource, or function.

persistence
The ability of a computerized system to remember the state of data or objects between runs.

plug-In
An external software or SQL program that is accessed by a larger application to provide added and customer-specific functionality.

port In programming, a port is a “logical connection place” and specifically, using the Internet’s protocol, TCP/IP, the way a client program specifies a particular server program on a computer in a network. When a service (server program) initially is started, it is said to bind to its designed port number. As any client program wants to use that server, it also must request to bind to the designated port number.

portal A Web site that offers users access to a broad array of resources and services, such as email, forums, search engines, and online shopping malls.

point-to-point protocol (PPP) A protocol for communication between two computers using a serial interface, typically a personal computer connected by a telephone line to a server.

process A process is an instance of a program running in a computer. It is close in meaning to task, a term used in some operating systems. In UNIX and some other operating systems, a process is started when a program is initiated (either by a user entering a shell command or by another program). Like a task, a process is a running program with which a particular set of data is associated so that the process can be kept track of. A process can initiate a sub-process, which is a child process (and the initiating process is sometimes referred to as its parent). A child process is a replica of the parent process and shares some of its resources, but cannot exist if the parent is terminated.

process management The EC Gateway module which allows definition of how the EDI solution will function in its native lights-out unattended mode of operation.

property A set of rules that govern the behavior of the computers communicating on a network.

protocol A set of rules that govern the transmission and reception of data.

put A request to store a message in a queue. Compare to get.

qualifier Answers the question “What is…?” It acts as a modifier.

queue A list constructed and maintained so that the next data element to be retrieved is the one stored first.

For example, one application can put a message on a queue, and another application can retrieve the message from the same queue.

record Basic building block of a database. Each record is the lowest-level complete entity within a table in the database. Also known as a “row.” A group of one or more records make up a table.

registry The part of the Window NT operating system that holds configuration information for a particular machine.

regular expression Strings that express rules for string pattern matching.

relational database A collection of data in which relationships between data items are explicitly specified as equally accessible attributes. The data is viewed as being stored in tables consisting of columns (data items) and rows (units of information). Relational databases can be accessed by SQL requests. See also Structured Query Language.

release character A character in data which indicates that a delimiter is following, and that the delimiter should be processed as data vice as a delimiter.

remote function call (RFC) An RFC is used to provide a “handshake” between two systems that are not connected. ALE and RFC are often used together.

remote systems management A feature that allows a System Administrator to manage multiple DirectConnect Servers and multiple services from a client.

repeating component A component, a field or format, that may appear multiple times in an input or output message.

request One or more database operations an application sends as a unit to the database. During a request, the application gives up control to the DBMS and waits for a response. See also commit, rollback, and unit of work.

RFC client A program that calls a remote function in an R/3 system using the RFC API.

RFC server A program that implements an ABAP function that can be called by an RFC client.

rollback An instruction to a database not to implement the changes requested in a unit of work and to return to the pre-transaction state. See also transaction and unit of work. Compare with commit.

rule A rule is uniquely defined by its application group, message type, and rule name. It contains evaluation criteria (a rules expression) and is associated with subscriptions to perform if the rule evaluates to true. Rules also have permissions that determine user access.

scalability The ability of an information system to provide high performance as greater demands are placed upon it, through the addition of extra computing power.

segment (EDI) Basic building block of an X12 transaction. Each segment is identified by a three-character code. Segments are comprised of elements and sub-elements and a group of segments are called a transaction or document.

segment (EMQ) An indexing device within a queue section. A series of contiguous slots that store information.

segment delimiter The character that appears at the end of each segment within an X12 transaction. For inbound X12 documents, this character definition is read directly from the envelope. For outbound documents, customer-specific segment delimiters are retrieved from the trading partner database.

serialize Process of taking items from a buffer and creating the corresponding tree representation from the wire format.

server A functional unit that provides shared services to workstations over a network. See also client/server. Compare with client.

servlet A servlet is a small, persistent, low-level program that runs on a server. The term was coined in the context of the Java applet, a small program that is sent as a separate file along with a Web (HTML) page.

Some programs that access databases based on user input need to be on the server. These programs were most often implemented using a Common Gateway Interface (CGI) application. However, if a Java virtual machine is running in the server, servlets can be implemented in Java. A Java servlet can execute more quickly than a CGI application. Instead of creating a separate program process, each user request is invoked as a thread in a single daemon process, so that the system overhead for each request is slight.

session A connection between two programs or processes. In APPC communications, sessions allow transaction programs to have conversations between the partner logical units. Sessions are established through SNA bind requests. There are several types of sessions: single, multiple, and parallel. See also advanced program-to-program communications and parallel sessions.

shared subscription A subscription that is associated with more than one rule. This subscription will only be retrieved once by the Rules APIs even if multiple associated rules evaluate true.

shortcut menu A floating list of actions. to open a shortcut menu, click an object and hold down the right mouse button; the available actions depend on the object.

Siebel VB A programming language that is used to write event procedures (scripts), which are attached to object definitions.

Simple Mail Transfer Protocol (SMTP) Simple Mail Transfer Protocol is a TCP/IP protocol used in sending and receiving e-mail. However, since it is limited in its ability to queue messages at the receiving end, it is usually used with one or two other protocols, POP3 or IMAP, that let the user save messages in a server mailbox and download them periodically from the server. SMTP is the Internet's standard host-to-host mail transport protocol and traditionally operates over TCP port 25.

Simple Object Access Protocol (SOAP)

Simple Object Access Protocol provides a way for applications to communicate with each other over the Internet, independent of platform. Remote objects can give a program almost unlimited power over the Internet, but most firewalls block non-HTTP requests. SOAP, an XML-based protocol, gets around this limitation to provide intraprocess communication across machines.

In Enterprise Portal, the implementation of SOAP is intended to provide businesses with a way to expose corporate software functionality to their customers with minimal firewall constraints, platform dependencies or complex development implementations involving DCOM or CORBA.

SOAP was developed by Microsoft, DevelopMentor, and Userland Software and has been proposed to the Internet Engineering Task Force (IETF) as a standard.

snap-In

A software component that provides easy access and configuration of information from the framework of the Microsoft Management Console for Windows NT.

socket server

The socket is the method for accomplishing inter-process communication. What this means is a socket is used to allow one process to speak to another, very much like the telephone is used to allow one person to speak to another. A socket must be created to listen for connections. Sockets have the ability to queue incoming connection requests. The communication that occurs between the client and the server must be reliable. That is, no data can be dropped and it must arrive on the client side in the same order in which the server sent it.

standard markup language

Standard ML is a general-purpose programming language designed for large projects.

Structured Query Language (SQL)

A language developed by IBM to process data in a relational database. SQL is an industry standard.

sub-element

Basic building block of an X12 transaction. Some X12 elements are comprised of multiple entities. The sub-element separator separates these sub-elements.

sub-element separator

The character that appears at the end of each sub-element within an X12 transaction. For inbound X12 documents, this character definition is read directly from the envelope. For outbound documents, customer-specific sub-element separators are retrieved from the trading partner database.

subscription

A subscription is uniquely identified by its application group, message type, and subscription name. It contains actions with options and can be associated with one or more rules. Subscriptions also have permissions that determine user access to change the subscription definition.

synchronous In electronic messaging, a method of operation in which sender and receiver applications are tightly coupled and dependent. The receiver must answer the sender's message immediately with a well-defined response; the sender must wait for the receiver's response before proceeding to the next operation. Compare to asynchronous.

syntax The rules for using transactions and documents.

system administrator The person at a computer installation who designs, controls, and manages the use of the computer system.

systems management The process of initiating, configuring, monitoring, and adjusting applications on a system.

TA1 acknowledgement The TA1 is an Interchange Acknowledgement. A network provider uses it to report the status of processing a received interchange header and trailer or the non-delivery.

table A database remembers relationships between pieces of information by storing the information in tables. The columns and rows in each table define the relationships in a highly structured way. We can classify tables by function into two types: support tables and data tables. Most tables fit into only one category, but some can serve as both support and data tables.

A support table stores information that changes infrequently and functions as a list from which you make selections.

tag A set of bits or characters that identify various conditions about data in a file. In Formatter, a standard value indicating the field's name.

target A system, program, or device that interprets and replies to requests received from a source.

template A form, mold, or pattern used as a guide to making something. In programming, a template is a generic class or other unit of source code that can be used as the basis for unique units of code.

thin client Thin client refers to the Net PC or the network computer (NC), personal computers for businesses that are centrally-managed, configured with only essential equipment, and don't have CD-ROM players, diskette drives, or expansion. Since the idea is to limit such computers to essential applications, they tend to remain "thin" in terms of the client applications they include.

thread safe A component or service is termed thread safe if multiple instances can be run at the same time. For example, a service may be used by multiple components that are running concurrently in multiple enclaves. Each component must be able to invoke that service without sharing violations.

trace The process of recording the sequence in which the statements in a program are executed and, optionally, the values of the program variables used in the statements.

trading partner Vendor or other party with whom business is conducted.

TRAN-IDE The IDE tool used to develop Production Objects used by e-Biz Impact. Production Objects process incoming messages by filtering, mapping, and/or exploding messages for the required endpoint destination applications.

transaction An activity or request. Additions, changes, and deletions are typical transactions stored in a computer. An exchange between a program on a local system and a program on a remote system that accomplishes a particular action or results.

transaction (EDI) A group of ANSI ASC X12 segments within an ANSI ASC X12 envelope.

transaction management A method of handling electronic messaging in which only committed messages are sent, and only messages received and committed are considered delivered.

transaction processing A method of handling computer operations in which the operations take place immediately upon receipt of the processing request. Also called realtime or online processing. Compare to batch processing.

transaction set A complete business document such as an invoice, a purchase order, or a remittance.

translator A piece of software that converts data from one format to another, often with intermediate lookups, validations, and edits.

transport The entity that stores individual message; for example, a queue.

Uniform Resource Indentifier A compact string of characters for identifying an abstract or physical resource and provides a simple and extensible means for identifying resources. An example of an URI is a URL.

Uniform Resource Locator A subset of a URI, a URL is like a networked extension of the standard filename concept: you can point to a file in a directory, but that file and directory can exist on any machine on the network. They can also be served by any of several different methods. URLs can also point to queries, documents stored deep within databases, and so on.

unit of work One of more database operations grouped under a commit or rollback. A unit of work ends when an application commits or rolls back a series of request or when the application terminates. See also commit, rollback, and transaction.

user exit A user-written program that processes data that is being transferred through an SAP link.

view An alternative representation of data from one or more tables. A view can include all or some of the columns contained in the table or tables on which it is defined.

workflow Software used to automatically route events or work-items from one user or program to another. Workflow is synonymous with process flow, although traditionally has been used in the context of person-to-person information flows.

X12 1) The committee within ANSI which defines EDI standards for business-to-business communications. 2) The set of transactions and their specifications, as defined by ANSI ASC X12.

x-log Error messages that contain a history of process actions and errors. X-logs are 50,000 bytes each; when xlogs reach the 50,000 bytes limit, alternative xlogs are automatically created named xlog2, xlog3, etc.

Index

A

B

C

D

E

F

I

J

M

N

O

P

R

S

T

V

X