IBM WebSphere Transformation Extender

Pack for X12

Version 2.7

Note
Before using this information, be sure to read the general information in “Notices” on page 17.

30 June 2006

This edition of this document applies to IBM WebSphere Transformation Extender Pack for X12 Version 2.7; and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this document, email DTX_doc_feedback@us.ibm.com . We look forward to hearing from you.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Chapter 1. Introduction

The X12 type trees are used to validate and ultimately transform data that conforms to the ANSI's X12 standard, as described in the ANSI X12 specifications.

The X12 standard is primarily used in the United States of America. The X12 standard is maintained by the American National Standards Institute (ANSI) and the Data Interchange Standards Association (DISA). The specifications can be obtained from the www.disa.org website, with links for the Industry specific sub-standards.

Installation

These trees are available to download from the IBM ESD site, www-306.ibm.com/software/. You will need your **User ID** and **Password** to login.

Chapter 2. X12 type trees

The type trees referenced in this documentation can be found in the IBM WebSphere Transformation Extender Pack for X12.

Overview

The root type of each ANSI X12 tree is called **EDI**. Each type tree has an **ANSI**, **Interchange**, and **Transmission** category stemming from the **EDI** root type. For example, the following **ansi4020.mtt** type tree contains the ANSI X12 version 4020 and the **ansi4030.mtt** type tree contains the ANSI X12 version 4030.

ANSI

The following are subtypes of the **ANSI** category:

- **Control** type defines the ANSI control structure objects. The **Control** category is the same on each **ANSI** version type tree.
- **Funct'lGroup** type defines the functional groups for the version as well as the data objects for that particular version.
- **V** category defines the version-specific data objects. For example, **V4030** is the name of the category in the 4030-version type tree (**ansi4030.mtt**).

Interchange

The **Interchange** type contains the data types for an interchange.

The **Interchange** category contains **Inbound** and **Outbound** partitioned group subtypes.

Transmission

The **Transmission** type contains the data types for a transmission, made up of one or more interchanges.

The **Transmission** category contains **Inbound** and **Outbound** group subtypes.

The ANSI trees include both inbound and outbound types, where appropriate. In addition, types called **Partner** define a generic trading partner.

ANSI Category

The **ANSI** category has three subtypes: **Control**, **Funct'lGroup,** and a **V** (*version*) category.

Control sub-tree

The subtypes of the **Control** category are the same in each ANSI tree. These types are used in ANSI envelope segments. Expand the **Control** category, under **ANSI**, to see the control types.

The following are subtypes of **Control**.

- **Delimiter** Items used to define syntax objects.
- **Element** Items used as envelope segment components.
- **ISAPartnerInfo** Components of the ISA segment.
- **Segment** Envelope segments.
- **Variant** Definitions of envelopes in other standards.

Segment group

The **Segment** group type in the **Control** category represents the envelope segments. Envelope segments are used:

- To surround functional groups within an **ANSI X12** interchange, the **ISA** and **IEA** segments in **X12**.
- For interchange acknowledgments, the **TA1** segment.
- To specify interchange delivery and handling requirements, the **ISB** and **ISE** segments.

Each **Segment** envelope type begins with a set of letters that identifies the segment, for example **IEA**. These same set of letters (**IEA**) are also used in the Properties window of the Type Designer to define the value of the initiator.

ISA segment

The **ISA** segment is the first segment in an **ANSI X12 Interchange**. The **ISA** segment contains data that identifies trading partners. This segment also specifies the delimiters and terminator within the interchange. The segment does not tell you which X12 version data is contained in the interchange. This version information is in the **GS** segment, which is discussed in the ″Funct'lGroup Sub-tree″ section.

The type **ISA** has an **Inbound** and an **Outbound** subtype, each of which has a **Partner** subtype. The **Inbound ISA** type is used as a component of the **Inbound Interchange** type. The **Outbound ISA** type is used as a component of the **Outbound Interchange** type.

The second component of **ISA** is the type **ISAPartnerInfo** which has the following components:

- Auth'nInfoQual'r Element
- Auth'nInfo Element
- SecurityInfoQual'r Element
- SecurityInfo Element
- Sender InterchangeIDQual'r Element
- InterchangeSenderID Element
- Reciever InterchangeIDQual'r Element
- InterchangeRcv'rID Element
- InterchangeDate Element

The following is an example of **ISA** segment data:

```
ISA*00*TSI       *01*92511930  *01*ME             *12*BRADLEY
*970815*1732*U*00201*000000050*0*T*>
```

It should be noted that this **ISA** example wraps around on the page, but there is no CL/LF in the **ISA** segment.

In the data of an **ISA** segment, the first thing after the initiator **ISA** is the element delimiter, so the first component of the **ISA** type is **Element Delimiter**. The value of the delimiter can be any of a number of different values. In input data, after the delimiter value is found as the first component of the **ISA**, it is set as the delimiter value for the rest of segments in the entire interchange. Likewise, the **Composite Delimiter** is the third component of the **ISA**, and the terminator is defined as the last component of the **ISA**.

When mapping from EDI data, the syntax values can be mapped from the components of the ISA in the input. When mapping to EDI data, you can set the syntax values by assigning values to the components of the **ISA**.

Note: The default value for the delimiter is an asterisk *, and the default value for the terminator is a carriage return/linefeed <CR><LF>. If you want to test inbound data that does not contain the interchange envelope, the data must contain the default delimiter and terminator values, or the data will be invalid. If the delimiter and terminator are different from the defaults, change the defaults in your type tree.

The **IEA** segment is the last segment in an interchange. The **IEA** segment has the same interchange control number that is in the **ISA**. The **IEA** segment also has a count of the functional groups within the interchange.

Interchange acknowledgment

The **TA1** segment is used as an interchange acknowledgment. It can be used in an interchange, just before the functional groups begin. Typically used only by network providers, an interchange containing **TA1**s consists of the **TA1**s surrounded by the **ISA** and **IEA** segments; it does not typically contain any functional groups.

Control elements

Elements under the **Control** category are the items used as components of the group types under the **Control** category. For example, elements that appear in the **ISA**, **ISB, and TA1** segments are defined here. Expand the **Element** category to view an alphabetical list of the elements.

Most of the definitions of the **Element** subtypes conform to the interchange standard, not the version standard. The main difference is the interpretation of the `N0` notation used in X12. The `N0` notation in the interchange standard refers to an unsigned integer. In the version standard, the `N0` notation refers to an integer, possibly preceded by a negative sign.

For example, the **InterchangeCtrl#** and **#InclFunct'lGroups** elements are part of the interchange standard and are defined in the type tree as an unsigned integer. The **GroupCtrl# Element**, however, appears as part of a version standard, and is therefore defined as an ANSI implicit decimal with zero decimal places.

Variant category

Envelope segments and elements for UCS (Uniform Communication Standard) are found under the **Variant** category.

The **BG** segment appears at the beginning of a **UCS** interchange, and the **EG** segment appears at the end. Elements that appear in the **BG** and **EG** segments are subtypes of the **Element** type.

Delimiter types

The **Delimiter** types are syntax items. They are used as element delimiters, composite delimiters, and terminator delimiters in the ANSI segments.

The values of these syntax objects may be different in different interchanges. The **Element Delimiter** appears as the first component of the **ISA**. The **Composite Delimiter** appears as the third component of the **ISA**, following the **ISAPartnerInfo** data fields. The **Terminator** appears as the last component of the **ISA**. For information about delimiters specified as syntax items, see the Type Designer documentation.

After the values of the syntax objects have been set by the **ISA**, they are used for the rest of the segments in that interchange. For example, the **Element Delimiter** value of the **IEA** segment is specified as the item **Element Delimiter**. The delimiter value found in the **ISA** will be the same value in the **IEA**.

The following item window displays the Include restrictions list for the **Element Delimiter** type; they are the possible values of the delimiter between elements.

Note: If a trading partner uses a value that is not in the restriction list, you can add the new value to the list.

Funct'lGroup sub-tree

The types under the **Funct'lGroup** category represent functional groups in the given X12 version. To allow for partner-specific definitions, the functional group types are organized under the category **Partner**, representing functional groups exchanged with a generic trading partner. Under **Partner**, the types are then organized by **Inbound** and **Outbound**. Subtypes of **Inbound** and **Outbound** are identical. **Inbound** and **Outbound** both have a subtype named **F**, followed by the X12 version. For example, **F4030** is the name of the functional group type in the **ansi4030.mtt** type tree.

The first segment of a functional group is the **GS** segment. The **GS** segment includes information about trading partners, the X12 version of the data, and the types of transaction sets that are contained within the functional group.

Expand the **Funct'lGroup** category to see all the functional groups in the given X12 version. Each functional group type is named according to the kinds of transaction sets that are contained within it. There are two possibilities for the type name of a functional group: name the type the transaction set number or name the type after the value of the element.

- If only one kind of transaction set can appear within the functional group, its type name is the transaction set number. For example, the functional group containing purchase orders, the transaction set #850, can only contain purchase orders. It cannot contain any other kind of transaction set. Therefore, the name of the functional group type is **#850**.

The following group window displays the components of the functional group **#850** in the **ansi4030.mtt** type tree. Notice that the second component is a series of **#850** transaction sets.

- If there can be more than one kind of transaction set within a functional group, the type name of the functional group is the value of the **FunctionalIDCd Element**. For example, the **CF** functional group can contain both #844 and #849 transaction sets. The type name of the functional group, then, is the value of the **FunctionalIDCd Element**, which is **CF**. Each functional group has a unique value for the **FunctionalIDCd Element**.

Note: There is one exception to the naming convention above. One functional group has a **FunctionalIDCd Element** whose value is `IN`. This functional group may contain **Invoices**, the **#810**, and **Operating Expense Statements**, the **#819**. Because IN is a reserved word in the Design Studio, the type name cannot be IN, so its name is **IE**.

As more industries join the EDI community, the list of functional groups gets longer. When you make your own tree from an ANSI tree, delete the functional groups you have no interest in. For example, if you create your own tree from the **ansi4030.mtt** tree, delete the functional group types under **F4030**, under **Inbound**, that you are not currently receiving. Then, do the same for the **F4030** group under **Outbound**; delete the functional groups you are not sending out. Remember that you can always add them back later.

Version specific sub-tree

The types specific to the given ANSI version are found under the ANSI category. The name of the category where the types are organized begins with a **V** and is followed by the ANSI version. For example, the name of the type in the **ansi4030.mtt** tree is **V4030.**

Elements

The item subtypes of the **Element** category are the ANSI data elements. The names of these types are abbreviations of the element descriptions in the Data Interchange Standards Association (DISA) documentation of the ASC X12 standards.

To view an element's full description

1. Select the type in the type tree.
2. Right-click and select **Properties** from the context menu.

   The description of the X12 data element is located in the **Value** column of the **Description** property.

There are over 1000 element types in an ANSI tree. Although they are arranged alphabetically, you might want to use the Find command on the **Edit** menu to locate a particular element.

The following list summarizes how an ANSI type tree corresponds to the DISA documentation of the ANSI EDI standards:

- Element codes are specified as item restrictions in an ANSI tree.
- Different elements with the same X12 description are named by appending the X12 data element reference number to the end of the type name.
- Some element types have subtypes. These will not be found as data elements in the DISA documentation. **Element** subtypes were added if the same element was used more than once as a component of the same segment. For example, if a segment has two **Date Element** components, one might be an **Arrival Date Element** and the other a **Departure Date Element**.

XComposites and MComposites

A composite is a set of related data elements. The ANSI standards incorporated the first composite in version 3030. In an ANSI X12 tree, the X12 composite data structures are found under the **XComposite** category, where the X stands for **X12**.

In addition, other sets of elements that ANSI does not define as composites have been defined as composites. These are located under the **MComposite** category. **MComposites** were formed to identify repetitive patterns of data elements and to group elements that are logically related.

Defining the composites in this manner simplifies type definitions and mapping rules. In addition, it is easier to identify which elements are required when others are present (or absent, as the case may be).

For example, the third component of the **ACT** segment is **MComposite**:

The ANSI X12 documentation states the following rule about two elements of the ACT segment:

```
P0304 - If either ACT03 or ACT04 is present, then the other is required
```

(where **ACT03** is the **IDCdQual'r Element** and **ACT04** is the **IDCd Element**).

If the data elements were defined as components of the **ACT** segment, a complex component rule, equivalent to the ANSI rule above, would be necessary. Instead, the two elements were defined as components of an **IDCd MComposite**:

The **IDCd MComposite** is then a component of the **ACT** segment, and is optional with a range of (0:1). This means that if an **IDCd MComposite** is in the data, both of its components must exist. In addition, if there is no **IDCd MComposite** in the data, neither component will be there. Defining the data in this way meets the requirements of the ANSI standards and simplifies the type definition.

Another difference between **XComposite** types and **MComposite** types is their delimiters. The delimiter of an **XComposite** type is the **Composite Delimiter**, which appears as data in the ISA. The delimiter of an **MComposite** is the **Element Delimiter**.

Segments

A segment is a logical unit of information, like a data record. The **Segment** type is partitioned by initiator values, each segment begins with a unique 2 or 3 character initiator, which identifies the segment. For example, the **ACT** segment begins with the initiator **ACT**.

Because an **Element Delimiter** must follow the segment initiator value and precede the first data element, each segment is defined as prefix delimited by the **Element Delimiter** and has a terminator of **Terminator Delimiter**.

Components of a segment may be **Element** types, **MComposite** types and **XComposite** types. The segments are defined to comply with the rules specified in the X12 documentation.

Sets - transactions sets, loops and blocks

Under the **Set** category, you will find the transaction sets and the loops for the given ANSI version. To allow for partner-specific definitions, the transaction set types are organized under the category **Partner**. **Partner** subtypes are then organized by **Inbound** and **Outbound;** these categories are identical.

Transaction sets are organized under categories

Each transaction set appears under a category with the same name. The name of the category is the same as the name of the **Funct'lGroup** type that contains that particular transaction set. For example, the #102 transaction set is a subtype of the **#102** category, just as the **Funct'lGroup** category containing the #102 transaction set is called **#102**.

The #844 transaction set is contained in a functional group that can have both #844 and #849 transactions. The functional group type name is **CF**, so the actual transaction set #844 is found under the category named **CF**.

If there is only one kind of transaction set under a category, the transaction set type name is simply **Transaction**. If there is more than one kind of transaction set under a category, the type **Transaction** is partitioned and the separate transactions are subtypes of it. For example, the actual #844 transaction set has the name **#844** and is a subtype of **Transaction**.

The **ST Segment** is the first component of each transaction, and is used as an identifier, the value of the **TSIDCd Element** in the **ST Segment** tells what kind of transaction it is, in this case, **#844** or **#849**.

If a transaction set cannot be found by its transaction set number, use the Find command. For example, if you did not know that the **#844** transaction set type was under the **CF** category, you could find it by choosing **Find** from the **Edit** menu, and entering #844 in the **Find What** field.

Loops and blocks

The loops and blocks within a transaction set appear under the same category where the transaction set definition itself is located.

Loops are groups of two or more semantically related segments, though in an actual X12 transmission, a loop may appear as only the single loop beginning segment. Loops may also contain other loops as components.

X12 defines two types of loops: unbounded and bounded. In unbounded loops, the first data segment in the loop appears once and only once in each loop occurrence (marking the start of an occurrence). Bounded loops are similar, but have no restriction on which data segment begins a loop occurrence and require a loop start (LS) segment before the first loop occurrence and a loop end (LE) segment after the last loop occurrence.

The type name of a loop is **Loop** followed by the identifier of the first segment in that loop and, if defined for the transaction set, the numeric Loop ID value specified by X12. For example, X12 version 4010 defines two different loops in transaction set #124, which begin with segment **LM**, loop ids 3200 and 3320. These loops are named **LoopLM3200** and **LoopLM3320** respectively. If no numeric Loop

Id values are specified for a transaction set and the same segment ID begins more than one loop, the sequential position of each loop within the transaction set is used to create unique loop names.

For example, version 4010, transaction set **#151** has 3 different loops, which begin with segment **PBI**, named **LoopPBI1**, **LoopPBI2** and **LoopPBI3**.:hen there is more than one transaction set under a category, loop and block definitions for a specific transaction set are found under a category whose name begins with the functional group ID code, and is followed by the transaction set number. For example, the loops in transaction set **#844** are subtypes of category **CF844**.

All bounded loop definitions for a transaction set appear under a group named **Block**. All bounded loops begin with an **LS Segment**, followed by the loop definition for the loop data segments and then an **LE**

Interchange category

Double-click the **X12 Inbound Interchange** type to view the components of the interchange.

The following table lists and describes the components of the **X12 Inbound Interchange**.

Component
Description

ISA Segment
The first component of an interchange is the envelope header. The **ISA** segment tells you who sent the interchange or who you are sending one to. It also contains the objects that will be used within the interchange for segment terminators, segment delimiters, and composite delimiters.

ISB Segment
Allows the sender to request additional services from the service request handlers.

ISE Segment
Allows the sender to request additional services from the service request handlers.

TA1 Segment
Optional and is typically used by network providers. TA1 is an interchange acknowledgment.

Funct'lGroup
Represents the functional groups that are sent or received.

IEA Segment
The interchange envelope trailer.

The **ISB** and **ISE** segments are found only in ANSI versions 3050 and later. For example, you will not see the **ISB** and **ISE** segments as components of an interchange in the **ansi3040.mtt** tree.

The **X12 Outbound Interchange** type is very similar to the structure of the **X12 Inbound Interchange** type. The only difference is that **Inbound** has been replaced with **Outbound**.

Restart attribute on functional group

The **Funct'lGroup** component of an interchange has the restart attribute. If you receive functional groups from different departments within a trading partner, one department's good data will not be lost if another department's data is bad. You can remove this restart attribute if you would rather go directly to the next interchange if bad data occurs within any **Inbound Funct'lGroup**.

Component rule on IEA segment

The **IEA Segment** component of an interchange has the following component rule:

```
InterchangeCtrl# Element:$ =
InterchangeCtrl# Element:.:Partner Inbound ISA Segment Control
ANSI & #InclFunct'lGroups Element:$-COUNTABS (Inbound Partner
Funct'lGroup ANSI)
```

The component rule states that the **InterchangeCtrl# Element** of the **IEA Segment** must be the same as the **InterchangeCtrl# Element** of the **ISA Segment**. If an **Inbound Partner Funct'lGroup** is invalid and a restart occurs, the component rule will keep different interchanges from being mixed up with one another.

Transmission category

A transmission is made up of interchanges, so a **Transmission** type is made up of **Interchange** types. An **Inbound Transmission** type is made up of **Inbound Interchange** types. For example, an **X12 Inbound Transmission** type is made up of **X12 Inbound Interchange** types and a **Partner X12 Inbound Transmission** type is made up of **Partner X12 Inbound Interchange** types.

The type **Partner** represents a generic trading partner. If you are trading with different partners who use different subsets of the X12 standards, you may want to rename the **Partner** type and then add new types to define the additional trading partners.

The components of **Outbound Transmission** types are similar to those of **Inbound** types.

Restart attribute on an interchange

The **Interchange** component of a **Transmission** type has the restart attribute. On input data, the restart attribute is intended to filter network noise and to detect invalid interchanges. If you receive an invalid interchange from one partner, the restart allows you to process good data in other interchanges in the same transmission.

When you receive network messages and you know where they appear in an inbound communication file, you may want to include them as a component of an **Inbound Transmission** type. If you do not include them as data, they will be rejected as invalid interchanges if they are not recognized as part of a transmission. However, the processing time will take longer than if you included these network messages as data.

For EDI output data, restart is useful for auditing data. You can audit the output data to report when outbound interchanges within a transmission are invalid.

Mapping ANSI data

When you create a map for mapping ANSI data, you will create an input or output card to represent the ANSI data. When you specify the **TypeName** for the input card, you must specify a type under the **Transmission** category.

Select **Inbound Transmission**, or one of its subtypes, when your input is ANSI data. When sending ANSI data, select **Outbound Transmission**, or one of its subtypes.

Input card - receiving ANSI data from multiple partners

When the input consists of only one interchange, select an **Interchange** type corresponding to the given **Transmission** type in the following table:

If You Receive
Select This Type

Multiple EDI standards
Inbound Transmission
Represents data from both ANSI and UCS.

Only ANSI X12 data, and you have defined different trading partners in your tree X12 Inbound Transmission

Only UCS data, and you have defined different trading partners in your tree
UCS Inbound Transmission

Only ANSI X12 data, and you have not defined different trading partners in your tree
Partner X12 Inbound Transmission

Only UCS data, and you have not defined different trading partners in your tree Partner UCS Inbound Transmission

See the IBM WebSphere Transformation Extender Packs for EDI documentation for further information about creating EDI type tree.

If you are familiar with Trading Partner PC, an Inbound Transmission (or one of its subtypes) defines the **mail_in_new** file. When mapping to EDI data, select an **Outbound Transmission** type for the output card. The type to choose depends on what you are sending.

Output card - sending ANSI data to multiple partners

When the output consists of only one interchange, select an **Interchange** type corresponding to the given **Transmission** type in the following table:

If You Are Sending
Select This Type

Multiple EDI standards
Outbound Transmission
Represents data from both ANSI and UCS.

Only ANSI X12 data, and you have defined different trading partners in your tree X12 Outbound Transmission

Only UCS data, and you have defined different trading partners in your tree
UCS Outbound Transmission

Only ANSI X12 data, and you have not defined different trading partners in your tree
Partner X12 Outbound Transmission

Only UCS data, and you have not defined different trading partners in your tree Partner UCS Outbound Transmission

See the IBM WebSphere DataSage TX Pack for EDI documentation for further information on creating EDI type trees. If you are familiar with Trading Partner PC, an **Outbound Transmission** type (or one of its subtypes) defines the **mail_out.new** file.

Note: If you are familiar with Trading Partner PC, an **Outbound Transmission** type (or one of its subtypes) defines the **mail_out.new** file.

Chapter 3. Example files

This chapter lists and describes the type trees and maps installed with the IBM WebSphere Transformation Extender Pack for X12.

ANSI X12 example files

The following ANSI X12 map and type tree example files install in the *install_dir***packs\EDI_v***n.n****x12\examples** directory:

Files in this path	
examples\ansi • partner.mms • myedi.mtt • profile.mtt	Description
examples\ansi\ANSIACK\ • audit997.mms • anyx12fa.mtt • myaudit.mtt • profile.mtt • x12_gen.mtt • x12_rej.mtt • x12_val2.mtt	Template partner profile data file, map and type tree.
examples\ansi\ASNTOEDI\ • sendasn.mms • robapp.mtt	Creating an outbound EDI Advance Ship Notice (#856) with its nested hierarchical levels.
examples\ansi\EDITOBOL\ • editobol.mms • bol.mtt	Mapping an inbound EDI Motor Carrier Shipment Information (#204) to an application file.
examples\ansi\EDITOINV\ • in_inv.mms • flatinvc.mtt	How to map inbound ANSI X12 EDI invoices (#810) to a proprietary file format.
examples\ansi\EDITOINV\ REJECT\ • invrejct.mms	Mapping invalid data.
examples\ansi\POTOEDI\ • sendpos.mms • ronapp.mtt	Mapping outbound ANSI X12 purchase orders (#850) from a proprietary file format. Uses a file containing trading partner information to lookup ANSI EDI envelope addresses for mapping EDI data for different partners.
examples\ansi\SDQ\ • sdq.mms	A turn-around mapping, from an inbound purchase order (#850) to an outbound invoice (#810), illustrating the SDQ problem.

Files in this path	
examples\ansi\TRACK\ • purge.mms • trackack.mms • trackout.mms • track.mtt	Building an EDI document control system using maps.

ANSI X12 standard type trees

The type trees that install in the *install_dir***packs****EDI_v**n.n***x12****trees** directory are the ANSI X12 standard type trees.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

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

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, a nd represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

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

AIX
AIX 5L
AS/400
Ascential
Ascential DataStage
Ascential Enterprise Integration Suite
Ascential QualityStage
Ascential RTI
Ascential Software
Ascential
CICS
DataStage
DB2
DB2 Universal Database
developerWorks
Footprint
Hiperspace
IBM
the IBM logo
ibm.com
IMS
Informix
Lotus
Lotus Notes
MQSeries
MVS
OS/390
OS/400
Passport Advantage
Redbooks
RISC System/6000
Roma
S/390
System z
Trading Partner
Tivoli

WebSphere
z/Architecture
z/OS
zSeries

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

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

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

This product includes software developed by the Eclipse Project (http://www.eclipse.org/).

IBM WebSphere Transformation Extender Pack for X12, Version 2.7

Index

V

X

Printed in USA