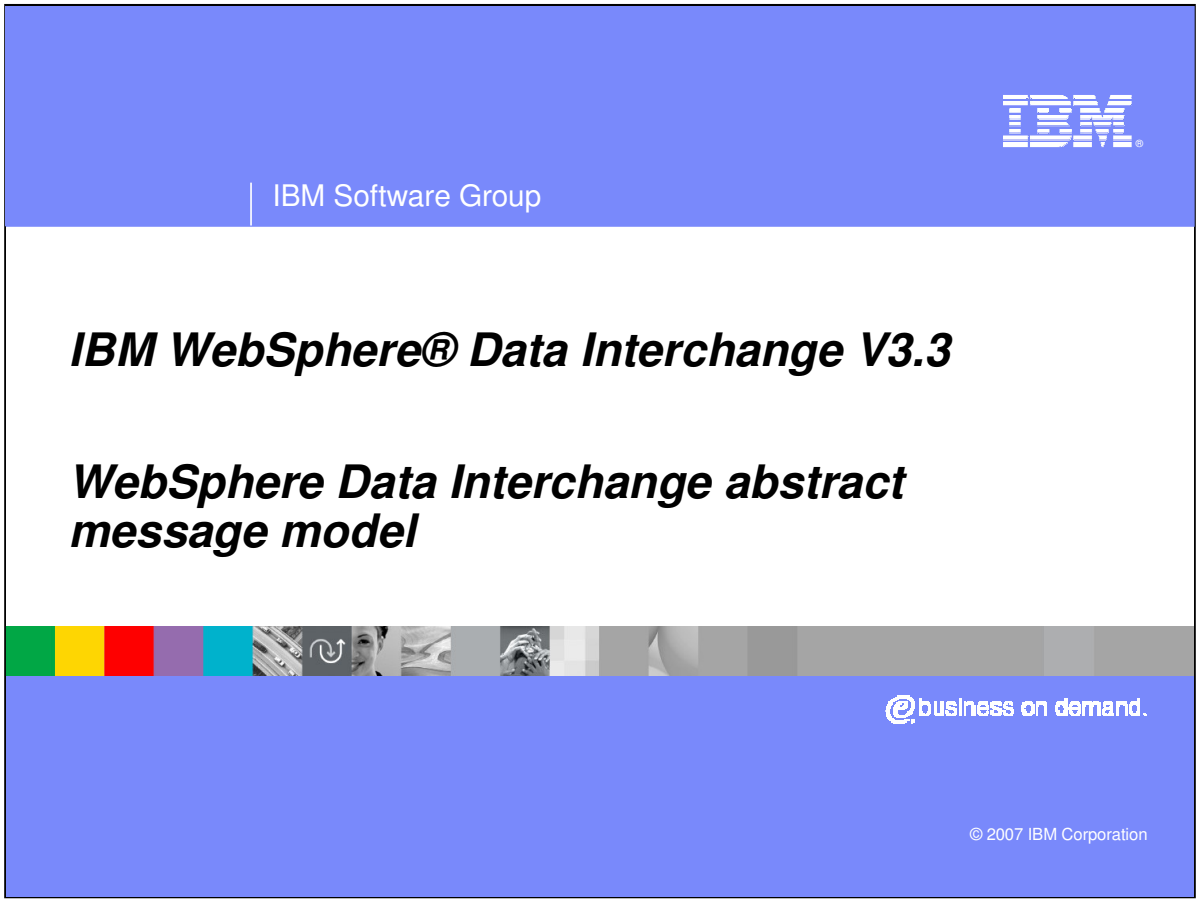


This presentation will describe the WebSphere Data Interchange abstract message model.

Agenda

- What is the abstract message model
- Why use abstract message model
- Abstract message model structure
- How does the abstract message model get created

The presentation will describe the WebSphere Data Interchange abstract message model used with the data transformation message flow.

What is the abstract message model

- The abstract message model.
  - All elements have a name.
  - Elements may be compound or simple.
  - The model of a message presented to a message processing node is that of a parse tree of syntax elements.
    - A name element has associated with it a string, which is the name of the element.
    - A value element has a value associate with it. Data types are character, real, integer, decimal, and byte array.
    - A name-value element is an optimization of the case where a name element contains only a value element and nothing else.
      - The element contains both a name and a value.

The Abstract Message Model (AMM) basically represents messages as a tree of elements. All elements have a name. Elements may be compound or simple. If an element has child element it is a compound element. If an element has no child elements, then it is a simple element and will have a type and a value.

The model of a message presented to a message processing node is that of a parse tree of syntax elements, each of which can be name elements, value elements, or name-value elements. Elements that contain a value will have a data type association for the type of data. The data types are character, real, integer, decimal, and byte array. WDI automatically converts value to these formats. For example, if the source value is data type Real the AMM data type would be decimal and if the source value id data type Binary the AMM data type would be byte array.

What is the abstract message model

- The root element is the unique element in the tree that has no parent.
- The root element is always a name element.
- Elements with the same parent element are siblings.
- Sibling elements have a definite order.
- Iterating over the set of children of an element will always present the elements in the same order.

The root element is unique and has no parent. Elements with the same parent are siblings. Sibling elements have a definite order, and iterating over the set of children of an element will always present the elements in the same order.

With EDI and data format source documents the AMM will be in the parsed order of the records, segments, and loops or the order of the data that was received. XML source documents will be in the order defined in the DTD or Schema. For example, if you have a DTD element definition <!ELEMENT Header (PONum, PODate, Sender, Receiver)>, the PONum, PODate, Sender, and Receiver elements would be children of the Header element.

With target documents the AMM will be constructed in the mapping order or the results of mapping execution. The target document definition or metadata used in the data transformation map is used to control the order.

Why abstract message model

- To support any-to-any translation.
- Send and receive translation is EDI based.
- The data transformation process is component based.
  - input messaging, input formatter (parser), translator, output formatter (generator), and output messaging as the components.
- Allows plug in capabilities with other IBM products like WebSphere MQ.

WebSphere Data Interchange needed to re-design the translation process. Send and Receive Translation is EDI based which makes supporting new technologies and data syntax impossible. For example XML. The Data Transformation process is component based, with Input Messaging, Input Formatter (Parser), Translator, Output Formatter (Generator), and Output Messaging as the components.

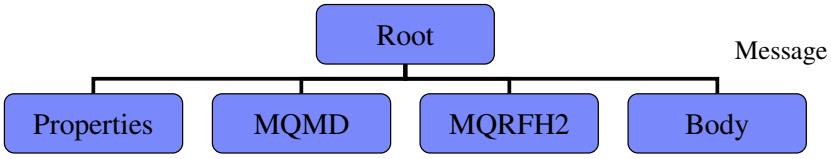
Why abstract message model

- The abstract message interface code, with functions to allow navigation from the current element to its:
  - Parent
  - First child
  - Last child
  - Previous (or left) sibling
  - Next (or right) sibling

The WebSphere Data Interchange Abstract Message interface code supports provides functions that enable a message processing node implementation to traverse the tree representation of the message. This allows for the manipulation of the elements with functions to create elements, to set or query their values, to insert new elements into the tree and to remove elements from the tree.

Abstract message structure

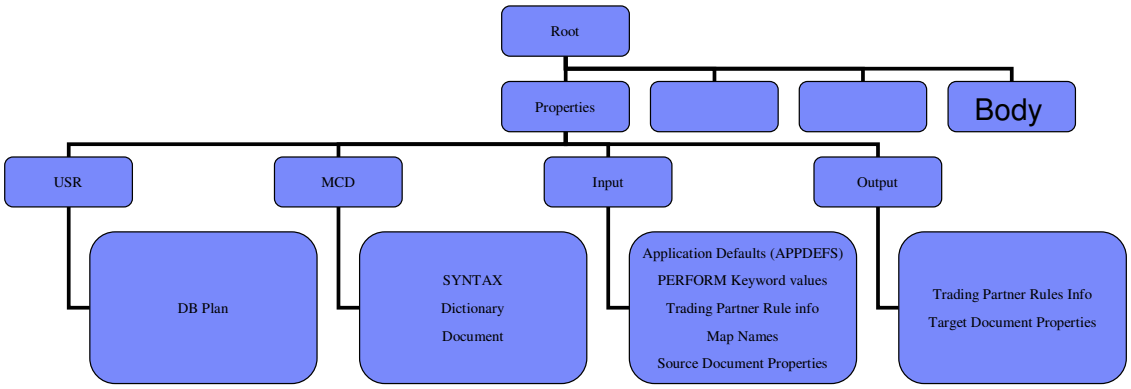
- The first generation of syntax elements of a typical message received by WebSphere MQ



The first generation of syntax elements of a typical message received by WebSphere MQ.

Abstract message structure

- Properties element



The WebSphere Data Interchange Abstract Message is within the Properties element. Database information is located in the USR element. The MCD element contains the syntax, dictionary, and document. The Input element contains initialization information, trading partner information, the data transformation map name, and source document properties. The Output element is similar to the Input element and contains target document properties and enveloping options.

Abstract message structure

- Properties element

```
ROOT (N)
...Properties (N)
.....USR (N)
.........DBSYSTEM (NV) = (char) ''
.........DBPLAN (NV) = (char) 'ediec32e'
.........DBOPEN (NV) = (char) 'Y'
.....mcd (N)
.........msd (NV) = (char) 'DataInterchange'
.........Set (NV) = (char) ‘XMLDICTIONARY'
.........Type (NV) = (char) ‘XMLDOCUMENT'
.........Fmt (NV) = (char) 'xml'
```

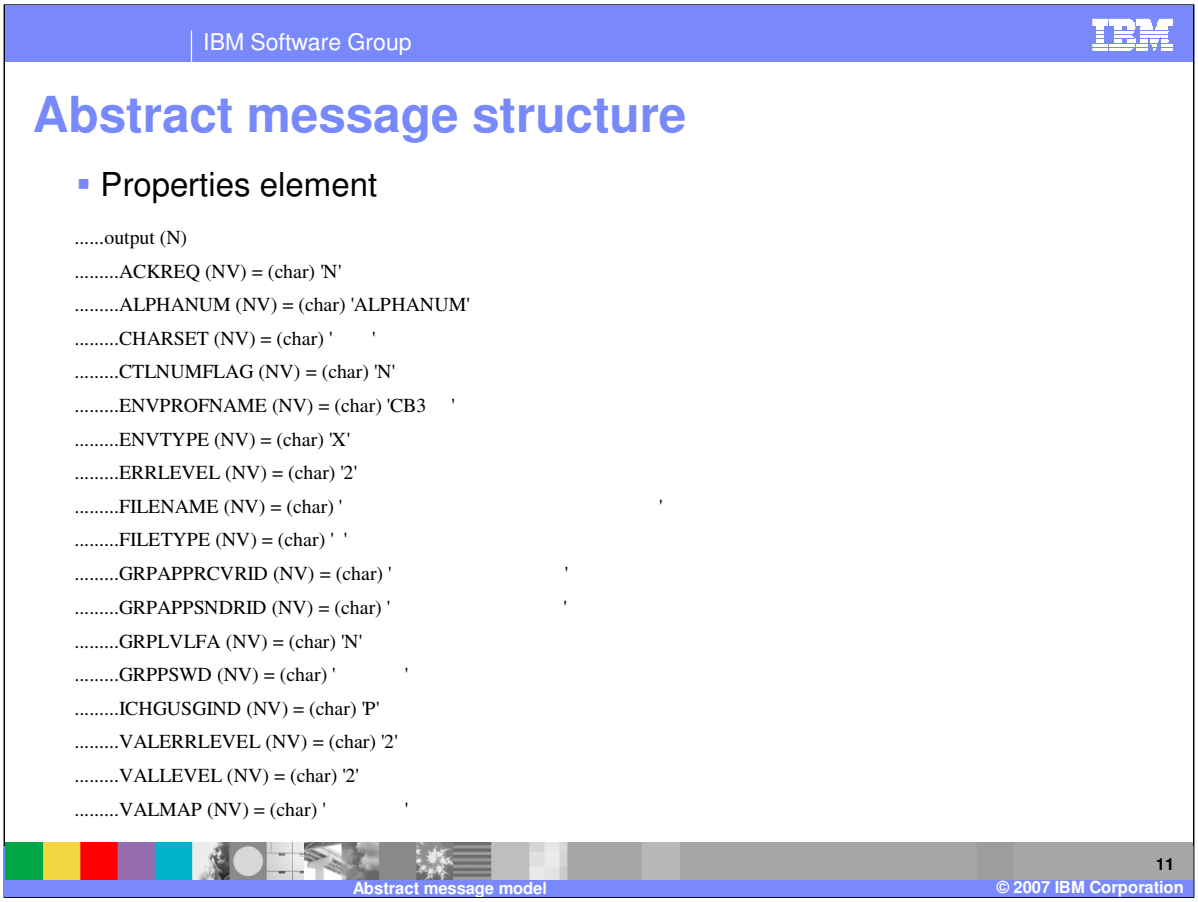

The properties element in the WebSphere Data Interchange Abstract Message format. This element contains information for processing for example the database plan and the metadata definition and type for the source data.

Abstract message structure

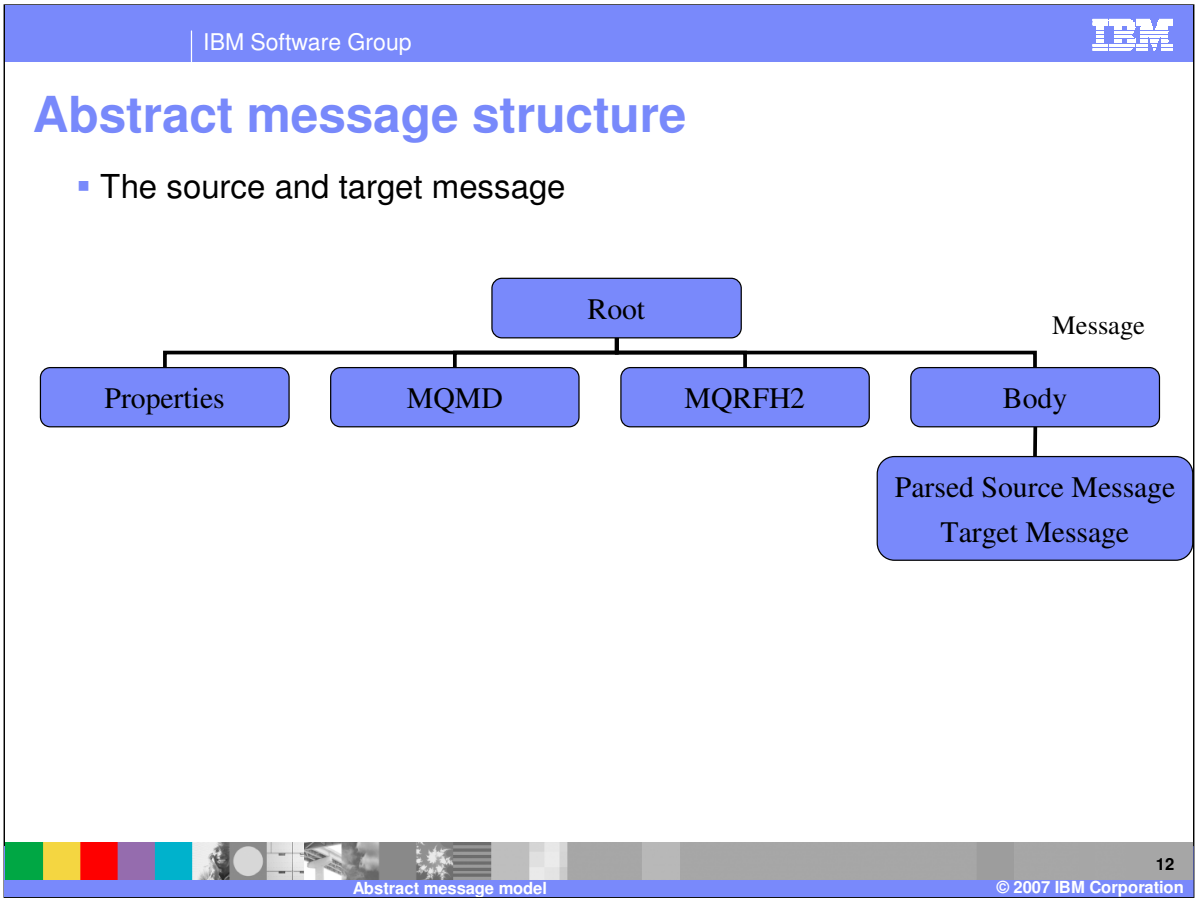
- Properties element

.....input (N)
.....CTRLYY (NV) = (char) '10'
.....APPLID (NV) = (char) 'EDIFFS'
.....TSACTIVE (NV) = (char) 'N'
.....TSIMAGE (NV) = (char) 'N'
.....TSFAIMAGE (NV) = (char) 'N'
.....SAPACTIVE (NV) = (char) 'N'
.....CDACTIVE (NV) = (char) 'N'
.....LASTMSG (NV) = (char) 'Y'
.....MSGSPLITCNT (NV) = (char) '0'
.....INPUTMSGCNT (NV) = (char) '1'
.....IGNOREINFO (NV) = (char) 'N'
.....IGNOREWARN (NV) = (char) 'N'
.....SERVSEGVAL (NV) = (char) '0'
.....TSPURGINT (NV) = (char) '30'

The input element in the WebSphere Data Interchange Abstract Message format. This element contains the source document properties and flags used for the translation process.



The output element in the WebSphere Data Interchange Abstract Message format. This element contains the target document properties and flags used for the translation process.



The parsed source abstract message is located in the Body element. After translation the target abstract message is located in the Body element.

Abstract message structure

...body (N)
.....XMLDOCUMENT (N)
.....GeneralMessageHeader (N)
.....ReceiverTPID (N)
.....No name (V) = (char) 'CB3'
.....Envelope (N)
.....Type (N)
.....No name (V) = (char) 'ADF'
.....ReceiverID (N)
.....Type (N)
.....No name (V) = (char) 'N/A'
.....Value (N)
.....No name (V) = (char) 'CB3DEU'
.....Date (N)
.....No name (V) = (char) '20031006'
.....Time (N)
.....No name (V) = (char) '170352'
.....Envelope (N)
.....Type (N)
.....No name (V) = (char) 'Audit'

<?xml version="1.0" encoding="UTF-8" ?>
<XMLDOCUMENT>
<GeneralMessageHeader>
<ReceiverTPID>**CB3**</ReceiverTPID>
<Envelope>
<Type>**ADF**</Type>
<ReceiverID>
<Type>**N/A**</Type>
<Value>**CB3DEU**</Value>
</ReceiverID>
<Date>**20031006**</Date>
<Time>**170352**</Time>
</Envelope>
<Envelope>
<Type>**Audit**</Type>
<Date>**20031006**</Date>
<Time>**170352**</Time>

This is a sample XML document on the right with the WebSphere Data Interchange Abstract Message on the left.

How does the abstract message get created

- The SYNTAX() PERFORM keyword tells the translation process the input message syntax
- The data transformation utility calls the logical message adapter for the syntax specified.
- The logical message adapter reads the data and parses out 1 logical message for the data transformation utility.
- The data transformation utility, passes the logical message to the WebSphere Data Interchange message broker.

The SYNTAX() PERFORM keyword tells the translation process the input message syntax. The Data Transformation (DT) Utility calls the Logical Message Adapter (LMA) for the Syntax specified. The Logical Message Adapter reads the data and parses out 1 logical message for the Data Transformation Utility. The Data Transformation Utility, passes the logical message to the WebSphere Data Interchange Message Broker (MB).

How does the abstract message get created

- The message broker begins construction on the abstract message
  - Sets the properties in the “input folder” under the properties folder,
  - Parses WebSphere MQ MQRFH2, MQMD headers
  - Creates the message flow for the translation (de-enveloper, validation, transformation)
  - Creates the body node and assigns a based on syntax
- When the de-enveloper request the message (body node), the parsing begins

The Message Broker begins construction on the Abstract Message. When a De-envelope request is made to get the body element, the parsing begins.

How does the abstract message get created

- The parser parses the logical message into the abstract form
  - EDI parser uses standard control string
    - A logical message is an interchange but this parser will parse out 1 transaction at a time
  - Data format parser uses the data format control string
  - XML parser uses the IBM XML tool kit to parse the XML input
    - The XML tool kit does the actual parsing and passes the information to the XML parser
    - The XML parser places the information into the AMM format

The Parser parses the logical message into the Abstract Message using the source metadata definition.

How does the abstract message get created

- The de-enveloper, passes the abstract message to the rules node and the map is identified
- After the rule is found, the translation is executed
- The translation takes the source abstract message and creates a target abstract message with mapping execution. It sets the target syntax
- After translation an enveloper node adds the envelope nodes to the abstract message
  - and sends the message back to the WebSphere Data Interchange message broker

The De-enveloper, passes the Abstract Message to the Rules lookup and the map is identified. After the Data Transformation Rule is found, the translation is executed. The translation takes the source Abstract Message and creates a target Abstract Message with mapping execution. It also sets the target Syntax.

After translation an enveloper node adds the envelope elements to the Abstract Message and sends the message back to the WebSphere Data Interchange Message Broker (MB).

How does the abstract message get created

- The message broker assigns a target serialization function based on the target syntax and the target abstract message
  - EDIWriteBuf
  - ADFWriteBuf
  - EDIEXWRT for XML
- The resulting buffer is passed back to the message broker and the message broker writes the data to output

The WDI Message Broker assigns a Target Serialization function based on the target Syntax and the target Abstract Message. The Abstract Message is serialized to a buffer. The resulting buffer is passed back to the Message Broker and the Message Broker writes the data to output.

Trademarks, copyrights, and disclaimers

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

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

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

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

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

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

Linux is a registered trademark of Linus Torvalds.

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

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

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

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

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

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

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