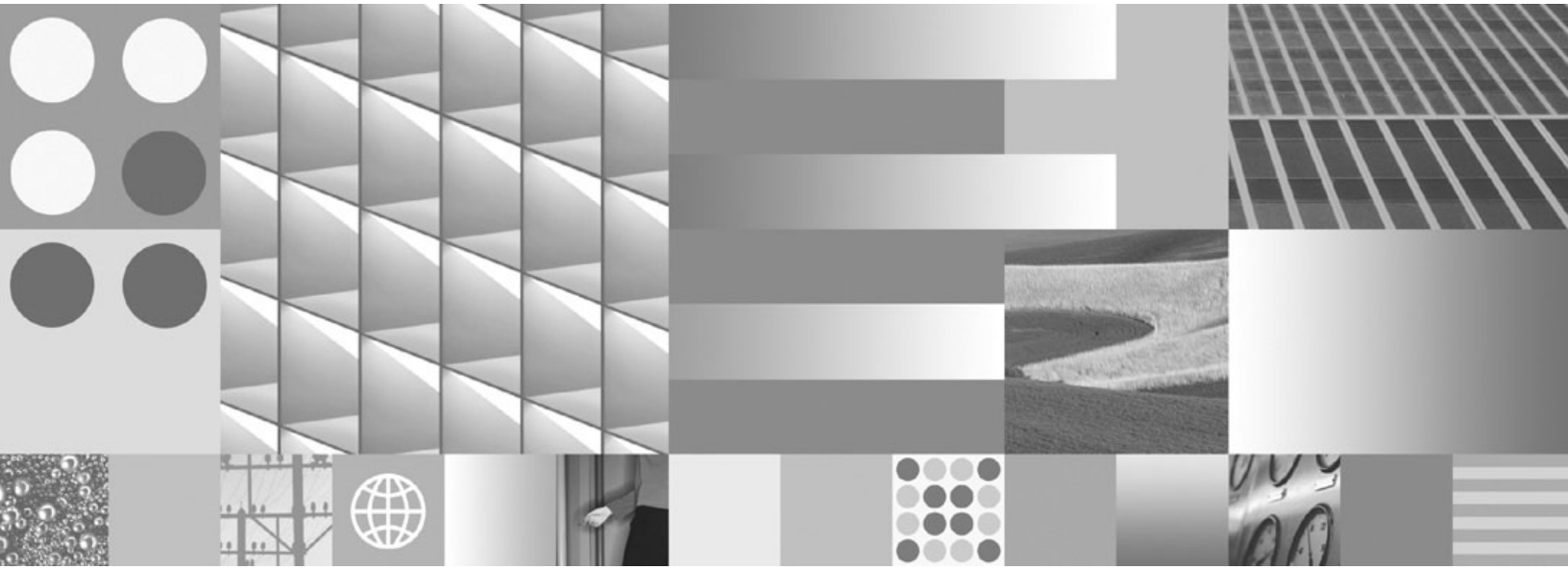
IBM Informix

Version 11.50

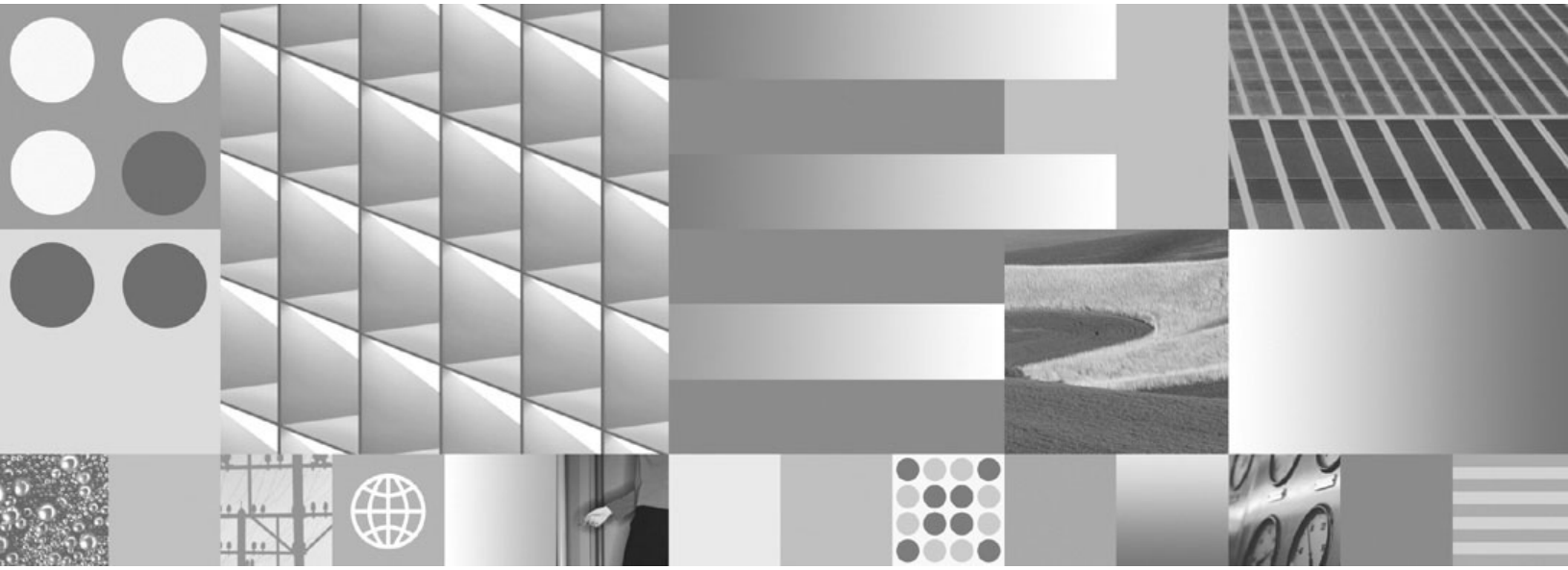


IBM Informix Dynamic Server XML User's Guide

SC23-9441-00

IBM Informix

Version 11.50



IBM Informix Dynamic Server XML User's Guide

SC23-9441-00

Note:
Before using this information and the product it supports read the information in “Notices” on page B-1.

Contents

Introduction

About This Publication

This publication includes information about using built-in functions for XML publishing with IBM Informix.

You should be familiar with the *IBM Informix Guide to SQL: Syntax*, which contains all the syntax descriptions for SQL and stored procedure language (SPL). The *IBM Informix Guide to SQL: Tutorial* shows how to use basic and advanced SQL and SPL routines to access and manipulate the data in your databases. The *IBM Informix Database Design and Implementation Guide* shows how to use SQL to implement and manage your databases.

See the documentation notes files for a list of the publications in the documentation set of your Informix® database server.

Types of Users

This publication is written for the following users:

- Database users
- Database administrators
- Database server administrators
- Database-application programmers

This publication assumes that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with computer programming and XML
- Some experience with database server administration, operating-system administration, or network administration

If you have limited experience with relational databases, SQL, or your operating system, refer to the *IBM Informix Dynamic Server Getting Started Guide* for your database server for a list of supplementary titles.

Assumptions About Your Locale

IBM Informix products can support many languages, cultures, and code sets. All the information related to character set, collation, and representation of numeric data, currency, date, and time is brought together in a single environment, called a Global Language Support (GLS) locale.

This publication assumes that your database uses the default locale. This default is **en_us.8859-1** (ISO 8859-1) on UNIX® platforms or **en_us.1252** (Microsoft® 1252) in Windows environments. This locale supports U.S. English format conventions for displaying and entering date, time, number, and currency values. It also supports the ISO 8859-1 (on UNIX and Linux®) or Microsoft 1252 (on Windows) code set, which includes the ASCII code set plus many 8-bit characters such as é, è, and ñ.

If you plan to use nondefault characters in your data or in SQL identifiers, or if you plan to use other collation rules for sorting character data, you need to specify the appropriate nondefault locale.

For instructions on how to specify a nondefault locale, and for additional syntax and other considerations related to GLS locales, see the *IBM Informix GLS User's Guide*.

What's New in XML User's Guide for Dynamic Server, Version 11.50

For a comprehensive list of new features for this release, see the *IBM Informix Dynamic Server Getting Started Guide*. The following changes and enhancements are relevant to this publication.

Table 1. What's New in IBM Informix Dynamic Server XML User's Guide

Overview	Reference
XSLT DataBlade® module This DataBlade module lets you call functions that run extensible stylesheet transformations (XSLT). These XSLTs can transform XML documents into a variety of output documents, including XML, HTML, and PDF.	See "XSLT Functions"

Documentation Conventions

This section describes the following conventions, which are used in the product documentation for IBM® Informix Dynamic Server:

- Typographical conventions
- Feature, product, and platform conventions
- Syntax diagrams
- Command-line conventions
- Example code conventions

Typographical Conventions

This publication uses the following conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.

Convention	Meaning
KEYWORD	Keywords of SQL, SPL, and some other programming languages appear in uppercase letters in a serif font.
italics	Within text, new terms and emphasized words appear in italics. Within syntax and code examples, variable values that you are to specify appear in italics.
boldface	Names of program entities (such as classes, events, and tables), environment variables, file names, path names, and interface elements (such as icons, menu items, and buttons) appear in boldface.
`monospace`	Information that the product displays and information that you enter appear in a monospace typeface.
`KEYSTROKE`	Keys that you are to press appear in uppercase letters in a sans serif font.
>	This symbol indicates a menu item. For example, "Choose **Tools > Options**" means choose the **Options** item from the **Tools** menu.

Feature, Product, and Platform Markup

Feature, product, and platform markup identifies paragraphs that contain feature-specific, product-specific, or platform-specific information. Some examples of this markup follow:

Dynamic Server

Identifies information that is specific to IBM Informix Dynamic Server

End of Dynamic Server

Windows Only

Identifies information that is specific to the Windows operating system

End of Windows Only

This markup can apply to one or more paragraphs within a section. When an entire section applies to a particular product or platform, this is noted as part of the heading text, for example:

Table Sorting (Windows)

Example Code Conventions

Examples of SQL code occur throughout this publication. Except as noted, the code is not specific to any single IBM Informix application development tool.

If only SQL statements are listed in the example, they are not delimited by semicolons. For instance, you might see the code in the following example:

```
CONNECT TO stores_demo
...

DELETE FROM customer
   WHERE customer_num = 121
```

```
...

COMMIT WORK
DISCONNECT CURRENT
```

To use this SQL code for a specific product, you must apply the syntax rules for that product. For example, if you are using an SQL API, you must use EXEC SQL at the start of each statement and a semicolon (or other appropriate delimiter) at the end of the statement. If you are using DB–Access, you must delimit multiple statements with semicolons.

Tip: Ellipsis points in a code example indicate that more code would be added in a full application, but it is not necessary to show it to describe the concept being discussed.

For detailed directions on using SQL statements for a particular application development tool or SQL API, see the documentation for your product.

Additional Documentation

You can view, search, and print all of the product documentation from the IBM Informix Dynamic Server information center on the Web at http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp.

For additional documentation about IBM Informix Dynamic Server and related products, including release notes, machine notes, and documentation notes, go to the online product library page at http://www.ibm.com/software/data/informix/pubs/library/. Alternatively, you can access or install the product documentation from the Quick Start CD that is shipped with the product.

Compliance with Industry Standards

The American National Standards Institute (ANSI) and the International Organization of Standardization (ISO) have jointly established a set of industry standards for the Structured Query Language (SQL). IBM Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992. In addition, many features of IBM Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL Common Applications Environment (CAE) standards.

Syntax Diagrams

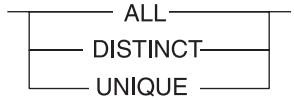
This guide uses syntax diagrams built with the following components to describe the syntax for statements and all commands other than system-level commands.

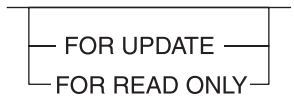
Table 2. Syntax Diagram Components

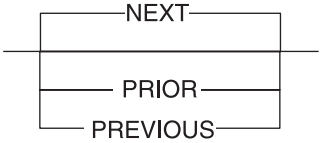
Component represented in PDF	Component represented in HTML	Meaning
	>>----------	Statement begins.
	---------->	Statement continues on next line.
	>----------	Statement continues from previous line.

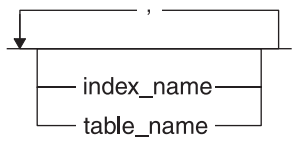
Table 2. Syntax Diagram Components (continued)

Component represented in PDF	Component represented in HTML	Meaning
	`----------><`	Statement ends.
	`--------SELECT----------`	Required item.
	`--+-----------------+---` `  '------LOCAL------'`	Optional item.

	`---+-----ALL-------+---`
	`   +--DISTINCT-----+`
	`   '---UNIQUE------'`
		Required item with choice. One and only one item must be present.

	`---+------------------+---`
	`    +--FOR UPDATE-----+`
	`    '--FOR READ ONLY--'`
		Optional items with choice are shown below the main line, one of which you might specify.

	`    .---NEXT---------.`
	`----+----------------+---`
	`    +---PRIOR--------+`
	`    '---PREVIOUS-----'`
		The values below the main line are optional, one of which you might specify. If you do not specify an item, the value above the line will be used as the default.

	`  .-------,-----------.`
	`  V                   |`
	`---+-----------------+---`
	`    +---index_name---+`
	`    '---table_name---'`
		Optional items. Several items are allowed; a comma must precede each repetition.

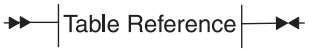
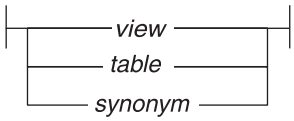
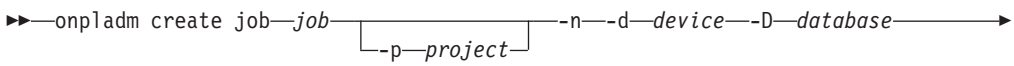
	`>>-| Table Reference |-><`	Reference to a syntax segment.

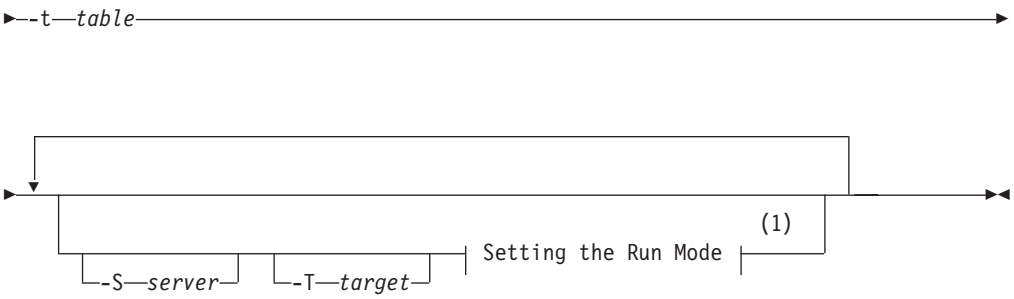
Table Reference
 
	`Table Reference`
	`|--+-----view--------+--|`
	`   +------table------+`
	`   '----synonym------'`
		Syntax segment.

How to Read a Command-Line Syntax Diagram

The following command-line syntax diagram uses some of the elements listed in the table in Syntax Diagrams.

Creating a No-Conversion Job



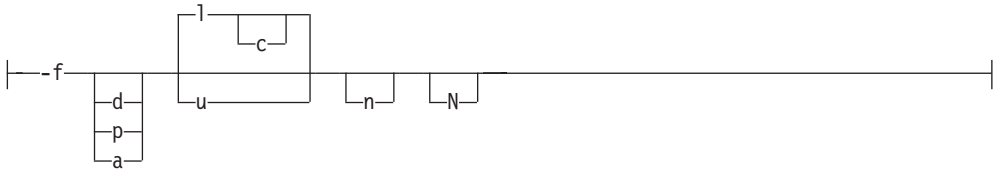


Notes:

1 See page Z-1

The second line in this diagram has a segment named "Setting the Run Mode," which according to the diagram footnote, is on page Z-1. If this was an actual cross-reference, you would find this segment in on the first page of Appendix Z. Instead, this segment is shown in the following segment diagram. Notice that the diagram uses segment start and end components.

Setting the Run Mode:



To see how to construct a command correctly, start at the top left of the main diagram. Follow the diagram to the right, including the elements that you want. The elements in this diagram are case sensitive because they illustrate utility syntax. Other types of syntax, such as SQL, are not case sensitive.

The Creating a No-Conversion Job diagram illustrates the following steps:

1. Type **onpladm create job** and then the name of the job.
2. Optionally, type **-p** and then the name of the project.
3. Type the following required elements:
   - **-n**
   - **-d** and the name of the device
   - **-D** and the name of the database
   - **-t** and the name of the table
4. Optionally, you can choose one or more of the following elements and repeat them an arbitrary number of times:
   - **-S** and the server name
   - **-T** and the target server name
   - The run mode. To set the run mode, follow the Setting the Run Mode segment diagram to type **-f**, optionally type **d**, **p**, or **a**, and then optionally type **l** or **u**.
5. Follow the diagram to the terminator.

Keywords and Punctuation

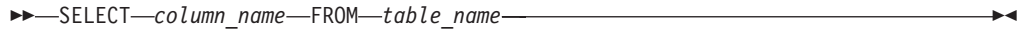
Keywords are words reserved for statements and all commands except system-level commands. When a keyword appears in a syntax diagram, it is shown in uppercase letters. When you use a keyword in a command, you can write it in uppercase or lowercase letters, but you must spell the keyword exactly as it appears in the syntax diagram.

You must also use any punctuation in your statements and commands exactly as shown in the syntax diagrams.

Identifiers and Names

Variables serve as placeholders for identifiers and names in the syntax diagrams and examples. You can replace a variable with an arbitrary name, identifier, or literal, depending on the context. Variables are also used to represent complex syntax elements that are expanded in additional syntax diagrams. When a variable appears in a syntax diagram, an example, or text, it is shown in *lowercase italic*.

The following syntax diagram uses variables to illustrate the general form of a simple SELECT statement.



When you write a SELECT statement of this form, you replace the variables *column_name* and *table_name* with the name of a specific column and table.

How to Provide Documentation Feedback

You are encouraged to send your comments about IBM Informix user documentation by using one of the following methods:

- Send e-mail to docinf@us.ibm.com.
- Go to the Information Center at http://publib.boulder.ibm.com/infocenter/idshelp/v115/index.jsp and open the topic that you want to comment on. Click **Feedback** at the bottom of the page, fill out the form, and submit your feedback.

Feedback from both methods is monitored by those who maintain the user documentation of Dynamic Server. The feedback methods are reserved for reporting errors and omissions in our documentation. For immediate help with a technical problem, contact IBM Technical Support. For instructions, see the IBM Informix Technical Support Web site at http://www.ibm.com/planetwide/.

We appreciate your suggestions.

Chapter 1. Creating the idsxmlvp Virtual Processor

The XML functions that Dynamic Server provides run in the virtual processor, **idsxmlvp**. You must start the virtual processor before you use the functions.

Start the virtual processor using one of the following methods, where *n* is the number of virtual processors to start:

- Add the following line to your ONCONFIG file:`VPCLASS idsxmlvp,num=n`
- As user informix, issue the following command: `onmode -p +n idsxmlvp`

Chapter 2. Publishing SQL Result Sets in XML

Several functions let you publish XML from SQL queries.

The XML functions that are provided in Dynamic Server are of two types: Those that return a maximum of LVARCHAR(32739), and those that return CLOB. All of the functions handle NULL values and special characters.

The input XML must include every value inside an element, not as an attribute. For example:

```
<employee>
  <givenname>Roy</givenname>
  <familyname>Connor</familyname>
  <address>
    <address1>123 First Street</address1>
    <city>Denver</city>
    <state/>CO</state>
    <zipcode>80111</zipcode>
  </address>
  <phone>303-555-1212</phone>
</employee>
```

The XML functions are summarized in Table 2-1.

Table 2-1. XML Publishing Functions

Action	Function	Comments
Return rows of SQL results as XML elements.	"genxml() and genxmlclob() XML Functions" on page 2-6	Similar to FOR XML RAW in Microsoft SQL Server
Return each column value as separate elements.	"genxmlelem() and genxmlelemclob() XML Functions" on page 2-7	Similar to FOR XML AUTO, ELEMENTS in Microsoft SQL Server
Return an XML schema and result in XML format.	"genxmlschema() and genxmlschemaclob() XML Functions" on page 2-10	Similar to FOR XML AUTO, XMLSCHEMA in Microsoft SQL Server
Returns the result set of a query in XML format.	"genxmlquery() and genxmlqueryclob() XML Functions" on page 2-9	These functions accept a SQL query as a parameter.
Returns the result set of a query in XML with the XML header.	"genxmlqueryhdr() and genxmlqueryhdrclob() XML Functions" on page 2-8	Every XML document must have a header. These functions provide a quick method of generating a header.
Evaluates an XPATH expression on a XML column, document, or string.	"extract() and extractclob() XML Functions" on page 2-3	Similar to the Oracle extract() function.
Returns the value of the XML node	"extractvalue() and extractvalueclob() XML Functions" on page 2-5	Similar to the Oracle extractvalue() function.
Verify whether a specific node exists in an XML document.	"existsnode() XML Function" on page 2-4	Similar to the Oracle exists() function.

Table 2-1. XML Publishing Functions (continued)

Action	Function	Comments
Parse an XML document to determine whether it is well formed.	"idsxmlparse() XML Function" on page 2-11	

XML Publishing

XML publishing provides a way to transform results of SQL queries into XML structures.

When you publish an XML document using the built-in XML publishing functions, you transform the result set of an SQL query into an XML structure, optionally including an XML schema and header. You can store the XML in Dynamic Server for use in XML-based applications.

The input XML must be in nested elements within an XML document, with each column being an XML element rather than an attribute. This format is sometimes called the FOR XML AUTO, ELEMENTS format.

The **genxmlschema** function and other XML publishing functions cannot publish BYTE or TEXT columns into an XML document. These functions take the input column as ROW types and then publish them. BYTE and TEXT are not allowed in ROW types, so cannot be used in these publishing functions.

ROW types are unsupported in distributed queries. These functions use ROW types. As a result, these publishing functions cannot be used in distributed queries or use a synonym referring to a non-local object.

Run the following statements on Dynamic Server before running the examples in this book:

```
CREATE DATABASE demo_xml IN datadbs with log;
CREATE TABLE tab (col2 lvarchar);
INSERT INTO tab VALUES ("
 <personnel>
  <person id="Jason.Ma">
  <name>
    <family>Ma</family>
    <given>Jason</given>
  </name>
  </person>
 </personnel>");
```

Special Characters in XML Functions

The XML functions of Dynamic Server automatically handle special characters.

When a SQL result set contains special characters, the XML function will automatically handle it. These special characters are listed in

Table 2-2. Special Characters Handled by XML Functions

Character	Resolved in XML as
Less than (<)	&lt;
Greater than (>)	&gt;
Double quote (″)	&quot;

Table 2-2. Special Characters Handled by XML Functions (continued)

Character	Resolved in XML as
Apostrophe (`)	'
Ampersand (&)	&amp;

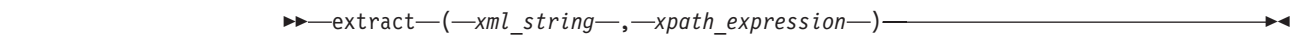
extract() and extractclob() XML Functions

Evaluates an XPATH expression on a XML column, document, or string. These functions are identical except that extractclob() returns a CLOB instead of LVARCHAR.

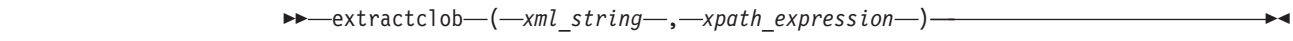
Purpose

Returns an XML fragment of the evaluated XML column, document, or string. For details on XPATH, see http://www.w3.org/TR/xpath.

extract() syntax



extractclob() syntax



Parameters

xml_string
The XML string or document to evaluate.

xpath_expression
An XPATH expression. For extract(), the string or document size cannot exceed 32739. For larger strings or documents, use extractclob().

Specify an absolute XPath_string with an initial slash. Omit the initial slash to indicate a path relative to the root node. If no match is found, these functions return an empty string.

Example 1

This example evaluates the XML contained in column `col2` of table `tab` and returns the given name for Jason Ma.

```
SELECT extract(col2, '/personnel/person[@id="Jason.Ma"]/name/given')
FROM tab;

<given>Jason</given>
```

Example 2

This example is similar to the first, except the entire name is returned.

```
SELECT extract(col2, '/personnel/person[@id="Jason.Ma"]/name')
FROM tab;
```

```
<name>
<family>Ma</family>
<given>Jason</given>
</name>
```

Example 3

In this example, only the second column contains XML.

```
SELECT warehouse_name, extract(warehouse_spec, '/Warehouse/Docks')::lvarchar(256)
"Number of Docks"
FROM warehouses
WHERE warehouse_spec IS NOT NULL;
```

```
WAREHOUSE_NAME           Number of Docks
-------------------      -------------------
Liverpool, England       <Docks>2</Docks>
Taipei, Taiwan           <Docks>1</Docks>
Buenos Aires, Argentina  <Docks>4</Docks>
Seattle, USA             <Docks>3</Docks>
```

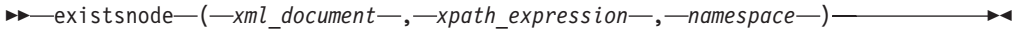
existsnode() XML Function

Determines whether the XPath evaluation results in at least one XML element. You can also specify a namespace to identify the mapping of prefixes that are specified in the XPath_string to the corresponding namespaces.

Purpose

Determines whether traversal of an XML document using a specified path results in any nodes. Returns 1 if one or more nodes are found; otherwise, returns 0.

existsnode() syntax



Parameters

xml_document
The XML document or fragment to evaluate. The document can be of type LVARCHAR or CLOB.

xpath_expression
The XPATH expression to search for XML nodes.

Specify an absolute XPath_string with an initial slash. Omit the initial slash to indicate a path relative to the root node. If no match is found, these functions return an empty string.

namespace
Use this parameter to identify the mapping of one or more prefixes that are specified in *xpath_expression* to the corresponding namespace.

Example 1

This example query returns a list of warehouse IDs and names for every warehouse that has an associated dock.

```
SELECT warehouse_id, warehouse_name
FROM warehouses
WHERE existsnode(warehouse_spec, '/Warehouse/Docks') = 1;
```

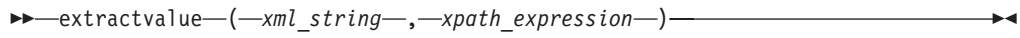
extractvalue() and extractvalueclob() XML Functions

Returns the value of the XML node in contrast to extract(), which returns the XML node.

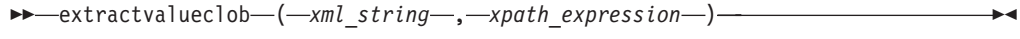
Purpose

Returns a value from evaluated XML column, document, or string. For details on XPATH, see http://www.w3.org/TR/xpath.

extractvalue() syntax



extractvalueclob() syntax



Parameters

xml_string
The XML string or document to evaluate.

xpath_expression
An XPATH expression. For extractvalue(), the string or document size cannot exceed 32739. For larger strings or documents, use extractvalueclob().

Specify an absolute XPath_string with an initial slash. Omit the initial slash to indicate a path relative to the root node. If no match is found, these functions return an empty string.

Example 1

This example returns the value given name of the person who is identified in the XPATH expression. No XML tags are returned.

```
SELECT extractvalue(col2, '/personnel/person[3]/name/given') FROM tab;
```

The output is the given name:

```
Jason
```

Example 2

This example returns the number of docks in several cities.

```
SELECT warehouse_name,
extractvalue(e.warehouse_spec, '/Warehouse/Docks')
"Docks"
FROM warehouses e
WHERE warehouse_spec IS NOT NULL;
```

```
WAREHOUSE_NAME          Docks
--------------------    ------------
Liverpool, England      2
Taipei, Taiwan          1
Buenos Aires, Argentina
Seattle, USA            3
```

genxml() and genxmlclob() XML Functions

Return rows of SQL results as XML elements. Use genxmlclob if the returned row is greater than LVARCHAR(32739).

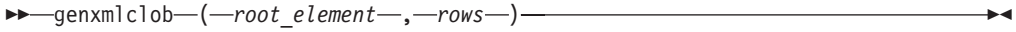
Purpose

Use these functions to create an XML row element for each row that is returned from an SQL query. Each column is an attribute of the row element. Use genxml for returned row values that are LVARCHAR(32739) or less. For larger values, use genxmlclob, which returns a CLOB.

These aggregate functions process the rows before an ORDER BY is completed. If order is important, use the derived table queries to get the result set in the correct order, and then apply the function on the result set. See "Enforcing order" on page 2-7 for details.

genxml() syntax

genxmlclob() syntax



Parameters

root_element
The table name or names of columns to return. To return all columns, specify the table name.

rows
The name given to the XML element of the returned row.

Example 1

This example shows how to retrieve XML rows from an SQL query on the following table:

Table 2-3. The classes table

classid	class	subject
1	125	Chemistry
2	250	Physics
3	375	Mathematics
4	500	Biology

The first parameter, *classes*, is the name of the table, which indicates to return all rows. The second parameter, *row*, is the name of the XML element that contains each returned row.

```
SELECT genxml(classes, "row") from classes;
```

The following lines show the results of the query in XML. The attributes in the rows are the names of the table columns.

```
<row classid="1" class="125" subject="Chemistry"/>
<row classid="2" class="250" subject="Physics"/>
<row classid="3" class="375" subject="Mathematics"/>
<row classid="4" class="500" subject="Biology"/>
```

Example 2

From the same table as Example 1, this example returns only the columns *classid* and *class*.

```
SELECT genxml(row(classid, class), "row") from classes;
```

```
<row classid="1" class="125" />
<row classid="2" class="250"/>
<row classid="3" class="375" />
<row classid="4" class="500" />
```

Example 3

This example uses genxmlclob() because a large result set is expected.

```
SELECT genxmlclob(row(Customers.Customid, Orders.Orderid,
       Customers.ContactName), "row")
From Customers, Orders
Where Customers.CustomerID = Orders.orderid;
```

This sample output shows only the first three rows:

```
<row Customerid="ALFKI" Orderid="10643" ContactName="Maria Anders"/>
<row Customerid="ALFKI" Orderid="10692" ContactName="Maria Anders"/>
<row Customerid="ALFKI" Orderid="10702" ContactName="Maria Anders"/>
.
.
.
```

Enforcing order

You can enforce the order of elements in XML document

```
SELECT genxml(row(c1, c2, c3), row)
FROM (SELECT a, b, c from t order by c, d)
AS vt(c1, c2, c3);
```

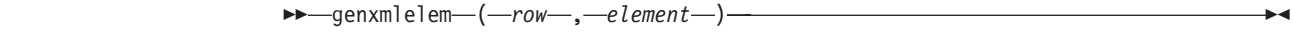
genxmlelem() and genxmlelemclob() XML Functions

These functions publish each element in the document separately.

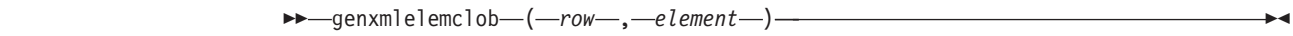
Purpose

These functions return each column value as separate elements, in contrast to genxml(), which returns column values as attributes of the row element.

genxmlelem() syntax



genxmlelemclob() syntax



Parameters

row
The rows and columns to return

element
The name of the element that contains the result set.

Example 1

This example uses the table Table 2-3 on page 2-6. The first parameter specifies the table name to retrieve all columns from the table. The second parameter specifies to place the output in an XML tag, *classes*.

```
SELECT genxmlelem(classes, "classes") from classes where classid = 1;
```

The query returns one row:

```
<classes>
<classid>1</classid>
<class>125</class>
<subject>Chemistry</subject>
</classes>
```

Example 2

This query returns a list of all employees from the employee table.

```
SELECT genxmlelemclob(employee, "employee") FROM employee;
```

```
<employee>
<givenname>Roy</givenname>
<familyname>Connor</familyname>
<address>
<address1>123 First Street</address1>
<city>Denver</city>
<state/>CO</state>
<zipcode>80111</zipcode>
</address>
<phone>303-555-1212</phone>
</employee>

.
.
.
```

genxmlqueryhdr() and genxmlqueryhdrclob() XML Functions

Returns the result set of a query in XML with the XML header.

Purpose

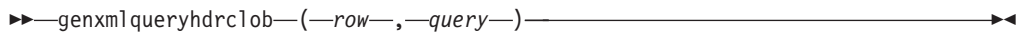
These functions are exactly the same as genxmlquery() and genxmlqueryclob(), except they include an XML header. An XML header specifies document properties such as the document encoding, the document type definition (DTD), and XML stylesheet (XSL). The following example shows a typical XML header:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Server SYSTEM "opt/pdos/etc/pdoslrd.dtd">
```

The encoding that is returned is the same as that of the operating system.

genxmlqueryhdr() syntax

genxmlqueryhdrclob() syntax



Parameters

row
The rows and columns to return

query
The SQL query whose result set will be returned as an XML document with a header.

Example 1

```
 EXECUTE FUNCTION genxmlqueryhdr('manufact_set','SELECT * FROM manufact');
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE manufact_set SYSTEM "../manufact_set.dtd">
<?xml-stylesheet type="text/xsl" href="../manufact_set.xsl" ?>
<manufact_set>
<row>
<manu_code>SMT</manu_code>
<manu_name>Smith          </manu_name>
<lead_time>3</lead_time>
</row>
<row>
<manu_code>ANZ</manu_code>
<manu_name>Anza           </manu_name>
<lead_time>5</lead_time>
</row>
<row>
<manu_code>NRG</manu_code>
<manu_name>Norge          </manu_name>
<lead_time>7</lead_time>
</row>
<row>
<manu_code>HSK</manu_code>
<manu_name>Husky          </manu_name>
<lead_time>5</lead_time>
</row>
</manufact_set>
```

genxmlquery() and genxmlqueryclob() XML Functions

These functions take a SQL query as a parameter and return the result set in XML.

Purpose

Use these functions to retrieve results with each column in an element.

genxmlquery() syntax

genxmlclobelemclob() syntax

Parameters

row
The rows and columns to return

query
The SQL query whose result set will be returned as XML.

Example 1

```
 EXECUTE FUNCTION genxmlquery('manufact_set','SELECT * FROM manufact');
<manufact_set>
<row>
  <manu_code>SMT</manu_code>
  <manu_name>Smith</manu_name>
  <lead_time>3</lead_time>
</row>
</manufact_set>
```

genxmlschema() and genxmlschemaclob() XML Functions

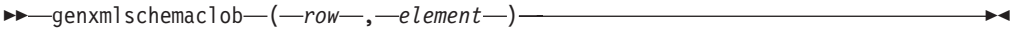
These functions generate an XML schema and result in XML format.

Purpose

These functions are identical to genxml() and genxmlclob(), but they also generate an XML schema.

genxmlschema() syntax

genxmlschemaclob() syntax



Parameters

row
The rows and columns to return

element
The name of the element that contains the result set.

Example 1

```
SELECT genxmlschema(customer, "customer")  FROM customers;
<?xml version="1.0" encoding="en_US.819" ?>
xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"\
targetNamespace="http://www.ibm.com"\
xmlns="http://www.ibm.com"\
ElementFormDefault="qualified">
<xs:element name="customer">
<xs:complexType>
```

```
<xs:sequence>
<xs:element name="customer_num" type="xs:serial"/>
<xs:element name="fname" type="xs:char(15)"/>
<xs:element name="lname" type="xs:char(15)"/>
<xs:element name="company" type="xs:char(20)"/>
<xs:element name="address1" type="xs:char(20)"/>
<xs:element name="city" type="xs:char(15)"/>
<xs:element name="state" type="xs:char(2)"/>
<xs:element name="zipcode" type="xs:char(5)"/>
<xs:element name="phone" type="xs:char(18)"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
<customer>
<row>
<customer_num>101</customer_num>
<fname>Ludwig </fname>
<lname>Pauli </lname>
<company>All Sports Supplies </company>
<address1>213 Erstwild Court </address1>
<city>Sunnyvale </city>
<state>CA</state>
<zipcode>94086</zipcode>
<phone>408-789-8075 </phone>
</row>
```

idsxmlparse() XML Function

Parse an XML document or fragment to determine whether it is well formed.

Purpose

This function returns an XML document or fragment if the input XML is well formed.

idsxmlparse() syntax

Parameters

xml_document
The XML document or fragment to parse to determine whether it is well formed.

Example 1

```
SELECT idsxmlparse(
'<purchaseOrder poNo="124356">
<customerName>ABC Enterprises</customerName>
<itemNo>F123456</itemNo>
</purchaseOrder>')
AS PO FROM systables where tabid = 1;
```

```
<purchaseOrder poNo="124356">
<customerName>ABC Enterprises</customerName>
<itemNo>F123456</itemNo>
</purchaseOrder>
```

Chapter 3. Transforming Documents with XSLT Functions

Apply XSL stylesheets and transforms to XML documents.

Extensible Stylesheet Language (XSL) uses XML elements to describe the format of a document. You can use the XSLT functions that IDS provides to apply XSL transformations (XSLT) to XML documents, resulting in a document in a different XML schema, HTML, PDF, or any defined type. XSL and XSLT are standards defined by the World Wide Web Consortium, which you can find at http://www.w3.org/TR/xslt.

IDS supports the XSLT version 1.0 standard for style sheets and uses the Xalan XSLT processor and the Xerces Java™ parser.

The documents take arguments of an XML file and a stylesheet. The output is a new document whose type is determined by the XSLT transform. The input arguments can be lvarchar, clob, or blob.

xsltransform Function

Use this function to return a document up to 32,739 bytes.

Purpose

The xsltransform function transforms an XML document with an XSL stylesheet and XSLT parameter. The returned document is of type LVARCHAR.

xsltransform Syntax

Parameters

xml_document
The XML document or fragment to transform. The document can be of type LVARCHAR, CLOB, or BLOB.

xsl_document
The XSL stylesheet document that is applied to the XML document. The XSL stylesheet can be of type LVARCHAR, CLOB, or BLOB.

Sample

Column **info** contains an XML document, and column **style** contains an XSL stylesheet:

Table 3-1. A row in table xmldocs

id	info	style
1	<?xml version='1.0' encoding='ISO-889–1' ?> <doc>Hello world!</doc>	<?xml version='1.0'?> <xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transform' version='1.0'> <xsl:output encoding='US-ASCII'/> <xsl:template match='doc'> <out> <xsl:value-of select='.'/> </out> </xsl:template> </xsl:stylesheet>

The XML document is transformed by the XSL stylesheet by calling xsltrasform:

```
select xsltransform(info, style) from xmldocs where id = 1
```

The following XML document is returned:

```
<?xml version='1.0' encoding="US-ASCII"?> <out>Hello world!</out>
```

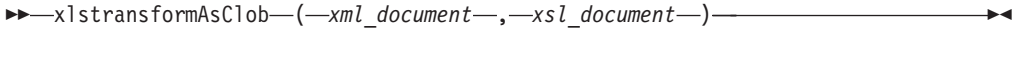
xsltransformAsClob Function

Use this function to return a CLOB document resulting from an XSL transform.

Purpose

The xsltransformAsClob function transforms an XML document with an XSL stylesheet and XSLT parameter. The returned document is of type CLOB.

xsltransformAsClob Syntax



Parameters

xml_document
The XML document or fragment to transform. The document can be of type LVARCHAR or CLOB.

xsl_document
The XSL stylesheet document that is applied to the XML document. The XSL stylesheet can be of type LVARCHAR or CLOB.

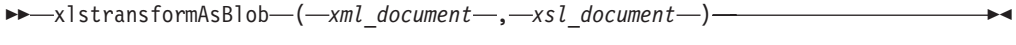
xsltransformAsBlob Function

Use this function to return a BLOB document resulting from an XSL transform.

Purpose

The xsltransformAsBlob function transforms an XML document with an XSL stylesheet and XSLT parameter. The returned document is of type BLOB.

xsltransformAsBlob Syntax



Parameters

xml_document
The XML document or fragment to transform. The document can be of type LVARCHAR or BLOB.

xsl_document
The XSL stylesheet document that is applied to the XML document. The XSL stylesheet can be of type LVARCHAR or BLOB.

Appendix. Accessibility

IBM strives to provide products with usable access for everyone, regardless of age or ability.

Accessibility features for IBM Informix Dynamic Server

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility Features

The following list includes the major accessibility features in IBM Informix Dynamic Server. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers.
- The attachment of alternative input and output devices.

Tip: The IBM Informix Dynamic Server Information Center and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard Navigation

This product uses standard Microsoft Windows® navigation keys.

Related Accessibility Information

IBM is committed to making our documentation accessible to persons with disabilities. Our publications are available in HTML format so that they can be accessed with assistive technology such as screen reader software. The syntax diagrams in our publications are available in dotted decimal format. For more information about the dotted decimal format, go to "Dotted Decimal Syntax Diagrams."

You can view the publications for IBM Informix Dynamic Server in Adobe Portable Document Format (PDF) using the Adobe Acrobat Reader.

IBM and Accessibility

See the *IBM Accessibility Center* at http://www.ibm.com/able for more information about the commitment that IBM has to accessibility.

Dotted Decimal Syntax Diagrams

The syntax diagrams in our publications are available in dotted decimal format, which is an accessible format that is available only if you are using a screen reader.

In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), the elements can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read punctuation. All syntax elements that have the same dotted decimal number (for example, all syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, the word or symbol is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is read as 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol that provides information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, this identifies a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should refer to a separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

? Specifies an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element (for example, 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! Specifies a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines

2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

* Specifies a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data-area, you know that you can include more than one data area or you can include none. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.
3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.

\+ Specifies a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times. For example, if you hear the line 6.1+ data-area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. As for the * symbol, you can only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.

Notices

IBM may not offer the products, services, or features discussed in this document in all 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 give 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
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
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 information 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, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

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. 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 IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

> © (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. (enter the year or years). All rights reserved.

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

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml.

Adobe, Acrobat, Portable Document Format (PDF), and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated 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.

Linux is a registered trademark of Linus Torvalds 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.

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

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

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

Index

A

D

I

S

V

X

Printed in USA

SC23-9441-00