Informix Product Family
Informix DataBlade Developers Kit
Version 4.20

IBM Informix DataBlade Module Development Overview

GC27-5555-00

Informix Product Family
Informix DataBlade Developers Kit
Version 4.20

IBM Informix DataBlade Module Development Overview

GC27-5555-00

Note
Before using this information and the product it supports, read the information in "Notices" on page D-1.

Contents

Introduction v
About this publication v
Types of users v
Software compatibility v
Example code conventions. v
Additional documentation vi
Compliance with industry standards vi
How to provide documentation feedback vi

Chapter 1. DataBlade module concepts 1-1
What are DataBlade modules? 1-1
Why extend your Informix database server? 1-1
Why create a DataBlade module? 1-3
Why use the DataBlade Developers Kit? 1-3
DataBlade modules and the database server 1-4
DataBlade module programming languages 1-5
Internal architecture of the database server 1-7
IBM Informix Client Software Development Kit 1-8
Client objects and programs 1-8
DataBlade module components 1-8
Aggregates 1-9
Data types 1-9
Routines 1-15
Casts 1-16
Interfaces 1-17
Errors. 1-17
Unit tests 1-18
Functional tests 1-18
Imported SQL files 1-18
Imported client files 1-18

Chapter 2. Build a DataBlade module 2-1
DataBlade Developers Kit tools 2-1
BladeSmith 2-1
BladePack 2-2
BladeManager 2-2
DBDK Visual C++ Add-In and IfxQuery 2-3
How to create a DataBlade module 2-4
DataBlade module development resources 2-5
The DataBlade Developers Kit Tutorial 2-5
Example DataBlade modules 2-6

Appendix A. DataBlade module documentation A-1
Publication overview A-1
Title-to-topic reference A-2
Topic-to-title reference A-3

Appendix B. Informix DataBlade modules B-1
IBM Informix Excalibur Text Search DataBlade Module B-1
Extensions to Informix B-1
IBM Informix Web DataBlade Module B-2
Extensions to Informix B-2

Appendix C. Accessibility C-1
Accessibility features for IBM Informix products C-1
Accessibility features C-1
Keyboard navigation C-1
Related accessibility information C-1
IBM and accessibility C-1
Dotted decimal syntax diagrams C-1

Notices D-1
Privacy policy considerations D-3
Trademarks D-3

Index X-1

Introduction

This introduction provides an overview of the information in this publication and describes the conventions it uses.

About this publication

This publication is an overview of the IBM® Informix® DataBlade® module development process. A DataBlade module extends the functionality of IBM Informix to handle data with user-defined routines or to handle nontraditional kinds of data, such as full text, images, video, spatial data, and time series.

Types of users

This publication servers as an overview for anyone interested in learning about DataBlade modules, including managers, developers who plan to create DataBlade modules, and developers who plan to create applications that use DataBlade modules. However, you should be familiar with SQL and basic programming concepts.

In contrast, the *IBM DataBlade Developers Kit User's Guide* provides technical information specifically for developers who are ready to develop DataBlade modules.

Software compatibility

For complete system requirements, see the *IBM Informix Read Me First* sheet for the IBM Informix DataBlade Developers Kit (DBDK).

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 that is 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

Documentation about this release of IBM Informix products is available in various formats.

You can access Informix technical information such as information centers, technotes, white papers, and IBM Redbooks® publications online at http://www.ibm.com/software/data/sw-library/.

Compliance with industry standards

IBM Informix products are compliant with various standards.

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.

How to provide documentation feedback

You are encouraged to send your comments about IBM Informix user documentation.

Use one of the following methods:

- Send email to docinf@us.ibm.com.
- In the Informix information center, which is available online at http://www.ibm.com/software/data/sw-library/, open the topic that you want to comment on. Click the feedback link at the bottom of the page, complete the form, and submit your feedback.
- Add comments to topics directly in the information center and read comments that were added by other users. Share information about the product documentation, participate in discussions with other users, rate topics, and more!

Feedback from all methods is monitored by the team that maintains the user documentation. The feedback methods are reserved for reporting errors and omissions in the documentation. For immediate help with a technical problem, contact IBM Technical Support at http://www.ibm.com/planetwide/.

We appreciate your suggestions.

Chapter 1. DataBlade module concepts

What are DataBlade modules?

A *DataBlade module* is a software package that extends the functionality of IBM Informix.

The package includes SQL statements and supporting code written in an external language or Informix SPL. DataBlade modules can also contain client components.

A DataBlade module adds user-defined *database objects* that extend the SQL syntax and commands you can use with your Informix database server. A database object is an SQL entity, such as a data type, routine, or database table. Your Informix database server handles DataBlade module objects as built-in objects. When it handles a user-defined database object, it executes the associated source code provided with the DataBlade module.

Extensions to your database server belong to two main categories:

Types of data
This category includes *extended data types* that are not built into the database server. Extended data types can contain multiple elements (row, collection, and opaque data types) and data types that support inheritance (distinct and row data types). The internal structure of opaque data types is not accessible through built-in SQL commands, but it can be accessed through user-defined routines and opaque data type support routines.

Routines
This category includes user-defined routines, aggregates, data type support routines, cast support routines, and routines that support user-defined access methods.

If you are unfamiliar with DataBlade module technology and the DataBlade Developers Kit, you might have the following questions:

- Why should I extend my database server?
- Why should I use a DataBlade module to extend my database server?
- Why should I use the DataBlade Developers Kit to create a DataBlade module?

Each of these questions is addressed in the following subsections.

Why extend your Informix database server?

The primary advantages of using the extensibility of IBM Informix over using traditional relational databases and applications are better performance, simpler applications, transaction control, and scalability.

Better performance

Your IBM Informix database server improves the performance of your applications in the following ways:

- User-defined routines are optimized.

  When you put your custom routines in the database server, the query optimizer can calculate when to run them during queries.
- Indexes increase query speed.

Indexes, created with secondary access methods, can efficiently find and compare values. Secondary access methods build and manipulate index structures on data. With your Informix database server, you can create indexes on data that cannot be sorted in a standard relational database. You can implement your data as extended data types and create functional indexes to speed sorting. A functional index sorts information about the data, instead of the data itself.

For example, if your data is images, you can index features of the images. Then, when you run a query to match an image, the index runs much faster than comparing the binary files of each image.

- Network traffic is reduced.

  When you use user-defined routines and other extensibility features, you perform more processing on the data within the database server. Therefore, you send less data to the client application.

Simpler applications

Using DataBlade modules simplifies applications in the following ways:

- DataBlade modules handle code for manipulating and storing data so the application does not have to.
- DataBlade module routines and data types can be accessed by using SQL.

  SQL is a standard language and does not require complex application code or programming languages.
- DataBlade modules are easy to upgrade

  When you change a DataBlade module, you do not need to relink existing applications; all changes are handled within the database server.
- All data is stored and processed in the same database server.

  For example, with a geospatial DataBlade module, geographic coordinates are analyzed and processed by the database server instead of in a complex application. In addition, the geospatial data is easily integrated with other types of data in a relational database.
- DataBlade modules are easy to combine.

  You can combine DataBlade modules that handle different kinds of data in the same database. You can then create one application to integrate all the data.

  For example, if a broadcast news company wanted to integrate video, images, audio, and text data for its programs, it could store all the data in one database and use a DataBlade module for each type of data. Then the company could use an application that employs the IBM Informix Web DataBlade Module to access information and display it in a web browser.

Transaction control

DataBlade modules become part of the database. Therefore, all operations carried out by DataBlade module routines are supported by database services, such as backup, rollback, and recovery. You can safely store your data, which you formerly stored in files, in the database by using smart large objects.

Scalability

DataBlade extensions to IBM Informix scale to many users and the database server itself.

Why create a DataBlade module?

You can extend your IBM Informix database server without creating a DataBlade module by executing the SQL statements to create each object individually. However, the advantages of packaging extended database objects in a DataBlade module are control and code reuse.

Control

DataBlade modules contain all related extended objects, enabling you to easily install, upgrade, and remove a whole module at once. If you need to fix a problem or add a feature to a program, you only have to do it in one place—the DataBlade module. Because a DataBlade module is a package ready to be distributed commercially or internally, these changes can be easily extended to any application that uses the DataBlade module. In addition, DataBlade modules make it easy for you to maintain version information about the software.

Code reuse

DataBlade modules can use the functionality of other DataBlade modules through *interfaces*. Interfaces are references to other DataBlade modules. When you include an interface in a DataBlade module, you create a dependency so that your DataBlade module can be used only if the DataBlade module that provides the interface is installed in the database server.

Some DataBlade modules are designed as *foundation* DataBlade modules. Foundation DataBlade modules are not usually intended to be used alone. For example, the IBM Informix Large Object Locator DataBlade Module handles the location of the large objects that other DataBlade modules use to store their data.

Why use the DataBlade Developers Kit?

Although you can create a DataBlade module manually, you can reduce development time considerably if you use the DataBlade Developers Kit.

Three graphical user interfaces are provided for DataBlade module development:

BladeSmith
Creates your DataBlade module

BladePack
Packages your DataBlade module

BladeManager
Makes your DataBlade module available in a database

In addition, the DataBlade Developers Kit provides the following tools for debugging your DataBlade module on Windows:

DBDK Visual C++ Add-In
Debugs your DataBlade module within Microsoft Visual C++

IfxQuery
Executes SQL debugging tests from within Microsoft Visual C++

The DataBlade Developers Kit reduces development time because it:

- uses wizards to guide you through complex SQL object creation options
- generates the following types of files:
  - Complete SQL definitions for your database objects

  - Complete code, or code entry points for C, C++, and Java™ source code
  - Unit tests for debugging user-defined routines, opaque data type support routines, and cast support functions
  - Functional tests for validating user-defined routines, opaque data type support routines, and cast support functions
- automates creating an interactive installation program for UNIX and Windows operating systems

The source code generated by the DataBlade Developers Kit follows good coding practices for your IBM Informix database server and ensures consistency among your user-defined routines.

DataBlade modules and the database server

This section describes the overall architecture of the IBM Informix database server, how DataBlade modules affect database server processes, and the application programming interfaces you can use in your DataBlade modules and client applications.

The following figure illustrates the following components of the Informix database server architecture when it includes DataBlade modules:

- DataBlade modules, which extend the capabilities of the database server
- DataBlade module application programming interfaces, which allow DataBlade modules access to data stored in a database
- The database server, which includes virtual processors that your Informix database server uses to process tasks, the shared memory that these virtual processors use, and the Java virtual machine to process routines written in Java
- The IBM Informix Client Software Development Kit, which includes client-side APIs that enable you to write client applications that access data stored in a database
- DataBlade module ActiveX and Java value objects, which enable you to provide client-side interfaces to extended data types and their support routines
- Client visualization tools, which enable you to view and manipulate data retrieved from DataBlade modules with third-party applications
- Client applications, which allow the user access to DataBlade module functions and data stored in a database

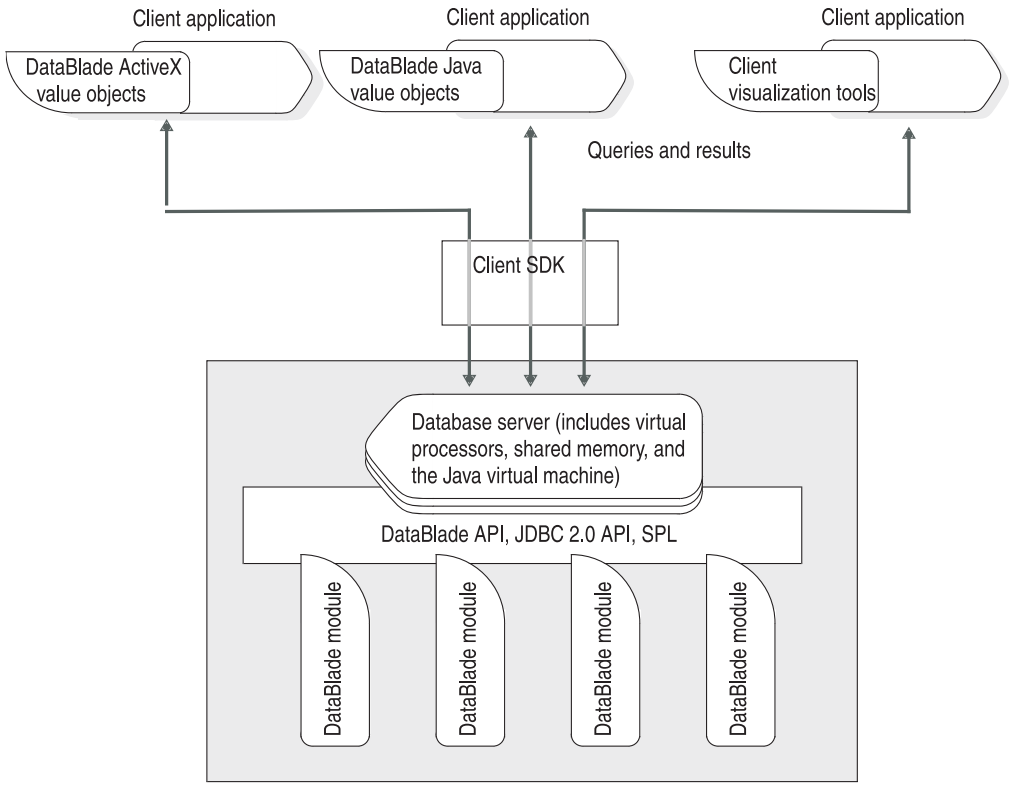

Figure 1-1. IBM Informix with DataBlade modules

The close integration of DataBlade modules with the database server means that the database server treats new, extended data types in the same way that it treats its own built-in data types.

Important: You must use the IBM Informix Dynamic Server with J/Foundation option of IBM Informix to enable services that use Java. For more information about J/Foundation, see the *J/Foundation Developer's Guide*.

DataBlade module programming languages

The DataBlade Developers Kit supports the following languages for programming DataBlade modules:

- C, using the DataBlade API
- C++, using the DataBlade API
- Java, using IBM Informix JDBC Driver
- Stored Procedure Language (SPL)

For more information about programming language options and restrictions, see the *IBM DataBlade Developers Kit User's Guide*.

C language

The IBM Informix DataBlade Developers Kit (DBDK) enables you to create database objects in C. You can create user-defined routines, cast support functions, aggregates, and opaque data type support routines in C.

The code generated for C by the DataBlade Developers Kit uses DataBlade API routines to communicate with the database server. The DataBlade API is the

primary API for the database server. The DataBlade API provides routines to manage database connections, send SQL command strings, process query results, manage database server events and errors, create database server routines, manage database server memory, and so on. The DataBlade API provides a subset of IBM Informix ESQL/C and IBM Informix GLS routines that you can use in your DataBlade module code. For more information about the DataBlade API, see the *IBM Informix DataBlade API Programmer's Guide*.

C++ language

The IBM Informix DataBlade Developers Kit (DBDK) currently allows you to write opaque data type support routines in C++.

You can also create ActiveX value objects to represent opaque data types on a client computer. If you want to include other database objects in your DataBlade module, the DataBlade Developers Kit allows you to code them in C or Java.

The C++ support routines use DataBlade API routines to process opaque data types in the database server. For more information about the DataBlade API, see the *IBM Informix DataBlade API Programmer's Guide*.

Important: It is recommended that developers create DataBlade modules in C++ only for client projects and for server projects that use Informix on Windows only.

Java language

The IBM Informix DataBlade Developers Kit (DBDK) enables you to create database objects in Java.

You can create user-defined routines, cast support functions, aggregates, and opaque data type support routines in Java. You can also create Java value objects to represent opaque data types on a client computer. You cannot create Java routines that take row or collection data types.

The code generated for Java by the DataBlade Developers Kit uses IBM Informix JDBC Driver methods to communicate with the database server. IBM Informix JDBC Driver supports the JDBC 2.0 API. You can use the JDBC 2.0 API to create database applications in Java.

For more information about IBM Informix JDBC Driver, see the *IBM Informix JDBC Driver Programmer's Guide*.

For a complete discussion about creating user-defined routines in Java, see the *J/Foundation Developer's Guide*.

Important: You must use the IBM Informix Dynamic Server with J/Foundation to upgrade IBM Informix to enable services that use Java. For more information about J/Foundation, see the *J/Foundation Developer's Guide*.

Informix Stored Procedure Language

You can use IBM Informix Stored Procedure Language (SPL) statements to write routines, and you can store these SPL routines in the database.

Informix SPL is an extension to SQL that provides flow control, such as looping and branching. SPL routines can execute routines written in C or other external languages, and external routines can execute SPL routines.

You can use SPL routines to perform any task that you can perform in SQL and to expand what you can accomplish with SQL alone. SPL routines are parsed and optimized when they are created. The DataBlade Developers Kit enables you to include SPL statements to create routines. For more information about SPL, see the *IBM Informix Guide to SQL: Tutorial*.

Internal architecture of the database server

If you want to add user-defined routines to your IBM Informix database server, you must understand the internal architecture of the database server and how DataBlade module routines can affect the database system.

The following aspects of the internal architecture of your Informix database server are affected the most by DataBlade modules:

- Virtual processors
- Memory management
- Java virtual machine

DataBlade modules and virtual processors

The internal architecture of your IBM Informix database server contains *virtual processors*.

Virtual processors are operating system tasks that execute requests. Virtual processors are separated into virtual processor classes. Each of the virtual processor classes provided in the database server handles a different type of task, such as executing queries and routines, fetching data from disk, and administering network connections. You can create user-defined virtual processors to handle tasks you define.

One of the critical virtual processors is the CPU VP, which acts as a router and handles basic administrative tasks, processes certain user requests, and delegates other requests to the appropriate processor. Tasks thus participate in a highly distributed environment that is optimized for performance and scalability.

By default, all user-defined routines execute in the CPU VP; however, if your DataBlade module routine uses certain system services, you must assign it to a user-defined virtual processor. A user-defined VP is created by the system administrator and executes only those routines assigned to it. For more information about the system services that require a user-defined VP, see the *IBM Informix Administrator's Guide*.

DataBlade module memory allocation

Another important aspect of the internal architecture of your IBM Informix database server is that virtual processors communicate with one another through shared memory. Therefore, when you write code for user-defined routines, you cannot use standard memory allocation functions. To manage memory for DataBlade modules, you must use the memory management functions provided by the DataBlade API or the JDBC 2.0 API.

See the *IBM Informix DataBlade API Programmer's Guide* or the *J/Foundation Developer's Guide* for complete information.

Java virtual machine

IBM Informix executes UDRs written in Java in its specialized virtual processors, called a *Java virtual processor* (JVP). A JVP embeds a Java virtual machine (JVM) in its code.

The JVPs are responsible for executing all UDRs written in Java. Although the JVPs are used for Java-related computation, they have the same capabilities as a user-defined VP, and they can process all types of SQL queries. This embedded VM architecture avoids the cost of shipping Java-related queries back and forth between CPU VPs and JVPs.

For more information about how the database server handles Java code, see the *J/Foundation Developer's Guide*.

IBM Informix Client Software Development Kit

The IBM Informix Client Software Development Kit (Client SDK) is a set of APIs you can use to develop applications for your Informix database server; they handle communication between the database server and the client application.

Client APIs allow you to write applications in the following languages:

- C
- Java
- ESQL/C

The Client SDK provides several connectivity products for ODBC-compliant applications and a global language support API.

For a list of current APIs, see the *IBM Informix Client Products Installation Guide*.

Client objects and programs

You can use the following types of client objects and programs with your DataBlade module applications:

ActiveX value objects
An ActiveX value object encapsulates data retrieved from an IBM Informix database server about an opaque type and its support routines for use by a client application. The DataBlade Developers Kit generates code for ActiveX value objects. You can use ActiveX value objects in a Microsoft Visual Basic program.

Java value objects
A Java value object encapsulates data retrieved from an IBM Informix database server about an opaque type and its support routines for use by a client application. The DataBlade Developers Kit generates code for Java value objects.

Client visualization tools
A visualization tool enables you to view and manipulate data retrieved by DataBlade modules with third-party applications.

DataBlade module components

You can include the following objects in the DataBlade module project you create with the DataBlade Developers Kit:

Aggregates
Perform user-defined computations on data

Data types
Characterize data to the database server (either built-in data types or new data types)

Routines
Operate on or return data

Casts Convert data from one type to another

Interfaces
Create dependencies between DataBlade modules

Errors Create messages raised by user-defined routines that appear as standard database server messages

Unit tests
Test your database objects during the coding and debugging cycle

Functional tests
Validate your completed DataBlade module routines

Imported SQL files
Include custom SQL statements to create tables, user-defined access methods, and other database objects in your DataBlade module

Imported client files
Include client components, such as query tools and ActiveX value objects, in your DataBlade module package

The DataBlade Developers Kit generates the SQL for each of the objects that you define or include.

Important: Not all database objects and options described in this section are available with all versions of IBM Informix. For more information, see the *IBM DataBlade Developers Kit User's Guide*.

Aggregates

An aggregate is a set of functions that returns information about a set of query results.

For example, the built-in SUM aggregate adds the values returned from a query and returns the result. An aggregate is invoked in SQL as a single function, but it is implemented as one or more support functions.

You can define two types of aggregates:

- Built-in aggregates whose support functions are overloaded to work with extended data types
- New, user-defined aggregates that implement user-defined routines.

The built-in aggregates are AVG, COUNT, MAX, MIN, SUM, RANGE, STDEV, and VARIANCE. The COUNT aggregate is defined for all data types. For more information about these aggregates, see the *IBM Informix Guide to SQL: Syntax*.

For more information about defining aggregates, see the *IBM DataBlade Developers Kit User's Guide*.

Data types

The database server uses data types to determine how to store and retrieve different kinds of data.

The following table lists the categories of data types available to you when you create a DataBlade module. You must define some of these data types; others you can use just as they are.

Data type	You define?	Code needed?	Description
Built-in	No	No	A native IBM Informix data type that comes with the database server
Qualified built-in	Yes	No	A built-in data type that takes one or more parameters, such as storage size, range of values, or precision
Built-in opaque	No	No (except for LVARCHAR)	A built-in data type that cannot be accessed in distributed queries by a different server instance.
User-defined opaque	Yes	Yes (the basic code needed to implement an opaque type is generated by BladeSmith)	A data type whose internal members cannot be accessed directly by using SQL statements
Distinct	Yes	No	A user-defined name for an existing built-in or extended type
Collection	Yes	No	A group of elements of the same data type
Row	Yes	No	A structured data type whose internal members can be directly accessed through SQL statements

An extended data type is a data type that is not built into IBM Informix. Extended data types include opaque data types, distinct data types, collection data types, and row data types. Extended data types are described in the *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Collection and row data types are extended data types built from a combination of other data types; their components are accessed through SQL statements.

Built-in data types

Built-in data types include character, numeric, time, large object, and Boolean data types. You can use built-in data types as building blocks in opaque, distinct, row, and collection data types.

Built-in data types are automatically included in your DataBlade module project file as imported objects.

For a complete list and descriptions of built-in data types, see the *IBM Informix Guide to SQL: Reference*.

Qualified built-in data types

A qualified data type is a built-in data type that has an added qualification that specifies information about the storage size, range of values, or precision of the type.

You must define a qualified data type by specifying its qualifications.

For example, the DECIMAL(*p*,*s*) data type can take qualifiers for precision (the total number of digits) and scale (the total number of digits following the decimal point). So, the DECIMAL(6,3) data type has six digits, with three digits to the right of the decimal point; for example, 123.456.

For a complete list of qualified data types and their parameters, see the *IBM Informix Guide to SQL: Reference*.

Distinct data types

A distinct data type is an existing data type to which you assign a unique name.

The distinct data type inherits all routines from the source data type, but it cannot be directly compared to the source data type without an explicit cast.

Use a distinct data type if you want to create routines that do not work on the source data type. You can use a distinct data type to control how the data type is cast, or converted, to other data types.

You can use distinct data types to create inheritance hierarchies, which allow you to write selective routines. A distinct data type can be passed to all routines defined for the source; however, the source data type cannot be passed to routines defined for the distinct data type.

Example

You can create two distinct data types based on the MONEY type: **lira** and **us_dollar**. To determine the dollar value of a specified lira value, you can create an explicit cast between **lira** and **us_dollar** by writing a function that multiplies the value of **lira** by the exchange rate. Using the cast allows you to perform arithmetic operations involving both distinct data types and to return a meaningful result.

For a description of distinct data types, see the *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Collection data types

A collection data type is a group of values of a single data type in a column. Each value is referred to as an element.

A collection data type is defined by using a type constructor and an element data type. Type constructors determine whether the database server checks for duplicate elements or orders the elements. The following table describes the collection type constructors.

Type constructor	Duplicates allowed?	Ordered?
SET	No	No
MULTISET	Yes	No
LIST	Yes	Yes

For a SET, the database server prevents insertion of duplicate elements. For a MULTISET, the database server takes no special actions. For a LIST, the database server orders the elements.

Elements can be almost any data type, including other extended data types and built-in data types such as smart large objects. You can access any element in a collection individually through SQL statements.

The number of elements in a collection is not mandated. You can change the number of elements in a collection without reinserting it into a table, and different rows can have different numbers of elements in their collections.

The following diagram illustrates a collection data type by using a SET constructor and the LVARCHAR data type in a column called **Dependents**.

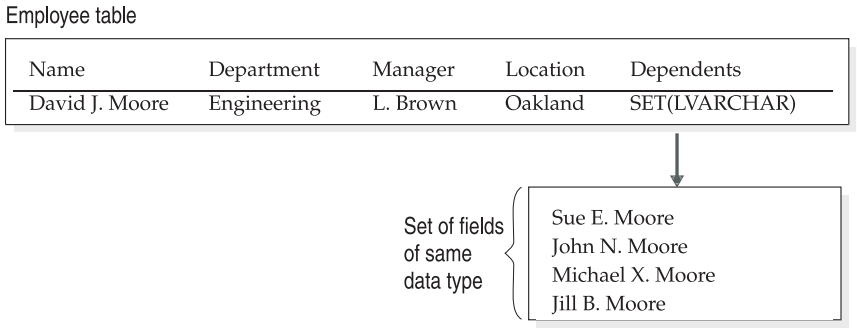

Figure 1-2. Sample collection data type

Instead of putting information about dependents in a separate table, all the information is contained in one row with collection data type. You can add or remove elements without altering the columns of the table.

You can use collection data types to reconfigure a table with awkwardly long rows by grouping data into a single column. Use a collection if you have data of the same data type that can be naturally grouped into a single column. You can group data even further by creating a collection of row types or other collections.

Collections are also useful as returned values: for example, a group of values from many rows in a column or fields in a row type. For example, if you want to obtain a list of every city in which your employees live from the sample collection data type in Figure 1-2, you could create a collection on the **Location** column to return a set of values.

The following function types can return collections:

- A user-defined function that returns a collection
- An iterator function that returns a single value at a time but is called repeatedly to assemble a collection

For a description of collection data types, see the *IBM Informix Guide to SQL: Tutorial*.

Row data types

A row data type can be thought of as a row of columns, of varying data types, stored in a single database table column.

Row data types follow essentially the same rules as database tables. The columns within a row data type are called fields. They can be almost any data type, including other extended data types and built-in data types, such as smart large objects. You can access fields individually by using SQL statements.

To create a row data type, you specify:

- a unique name for the whole row type
- a unique name for each field
- a data type for each field.

The following diagram illustrates a row type named **address_t** in a column named **Address**.

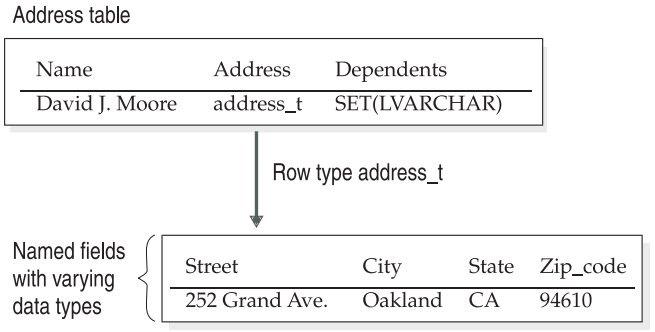

Figure 1-3. Sample row data type

Instead of having additional columns in the **Address** table, the row data type groups data that is most often accessed together in one column. The table **Address** consists of the columns **Name**(LVARCHAR(30)), **Address**(**address_t**), and **Dependents**(SET(LVARCHAR)). The row data type **address_t** consists of the named fields **Street**(LVARCHAR(20)), **City**(LVARCHAR(20)), **State**(CHAR(2)), and **Zip_code**(INTEGER).

Like collection data types, row data types allow you to reconfigure your database table. Use a row type if you have data of differing data types that group naturally into a single column. You can further group your data if you include a collection or another row data type as a field within your row data type.

Row data types can be useful for handling smart large objects. For example, if a row data type has a field that is an opaque data type containing an image in a smart large object, the other fields of the row data type could contain additional information about the image.

For best performance, use row data types if most user queries access all or most of the fields of the row data type.

You can use row data types to create inheritance hierarchies, allowing you to write selective routines. A child row data type inherits the fields of its parent and can be passed to all routines defined for the parent; however, the parent data type cannot be passed to routines defined for the child data type.

For a discussion about row data types, see the *IBM Informix Guide to SQL: Tutorial*.

Opaque data types

An opaque data type is a user-defined data structure.

To successfully interpret opaque data types, the database server requires that the DataBlade module provide opaque data type support routines. You must provide support routines for your opaque data type.

BladeSmith generates boilerplate code for opaque data type support routines. You can write additional code in C or Java to implement the functionality your opaque data type requires. If you provide ActiveX value objects as a client-side interface to your opaque data types, you can write the underlying support routines for the opaque data type in C++. See "Opaque data type support routines" on page 1-15 for more information.

Opaque data types typically contain more than one member, so they are similar to row data types, except that the database server can only operate on the individual components with support routines you define in the DataBlade module.

The following diagram illustrates an opaque data type called **circle**, based on a structure called **circle_t**, in a column called **circle_col**.

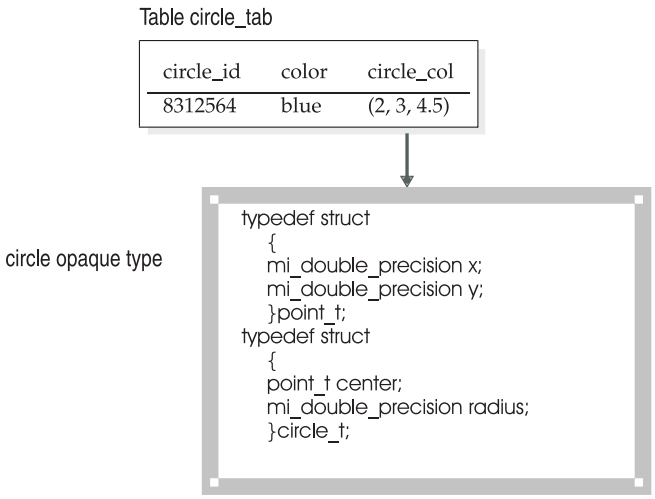

Figure 1-4. Sample opaque type

The table **circle_tab** consists of the columns **circle_id**(SERIAL), **color**(VARCHAR(15)), and **circle_col**(**circle**). The opaque data type **circle** is defined as a C structure, **circle_t**, which contains a **radius** member and another structure, **point_t**. The **point_t** structure contains **x** and **y** members. To the database server, however, the whole **circle_t** structure is indivisible, unless you provide accessor functions.

Use an opaque type to extend SQL to address fundamentally new data types and allow operations on them. Such data types are typically indivisible or made up of components to which you want to control access.

For example, geographic data and many sorts of rich media data (images, audio, text, and so on) are have been represented by the use of opaque types. Opaque data types often contain *smart large objects*. Smart large objects are logically stored in a column in a database table but physically stored in a file. For example, an opaque data type for images could have a smart large object member containing the image and other members containing metadata about the image.

Opaque types require more work to create than other data types because you must write all the routines that support them.

Opaque data type support routines:

BladeSmith enables you to generate support routine code for opaque data types. You might have to add code to implement the functionality your opaque data type requires.

The following table describes the support routines you can create and indicates the categories of opaque types for which they are recommended

Function	Recommended for	Description
Text input and output	All opaque types	Convert between external and internal representations.
Send and receive	All opaque types	Convert between internal representation on the database server and client computers. Not available for Java.
Text import and export	All opaque types	Process opaque types for bulk loading and unloading as textual data to and from a file.
Import binary and export binary	All opaque types	Process opaque types for bulk loading and unloading as binary data to and from a file. Not available for Java.
Assign() and **Destroy()**	Large objects and multi-representational types	Stores or deletes data on disk just before a commit: for example, to ensure proper reference counting on smart large objects. Not available for ActiveX.
LOhandles()	Large objects and multi-representational types	Returns the large object handle or list of large-object handles in opaque types that contain smart large objects. Not available for ActiveX.
Compare()	Opaque data types sorted by a B-tree or R-tree index	Sorts opaque type data within SQL statements and supports the B-tree and R-tree access method.
Statistics support	All opaque types	Compile information about the values in an opaque data type column that the optimizer can use to create a query plan. Not available for Java or ActiveX.

For a description of creating opaque types and their support routines, see the *IBM DataBlade Developers Kit User's Guide* or the *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Routines

A routine is a stored collection of programming statements that allows you to manipulate data.

A routine can be a function, which returns values, or a procedure, which does not. You can write routines in the IBM Informix Stored Procedure Language (SPL), or in an external language, such as C or Java.

Important: Not all routine options are available for SPL and Java. For more information, see the *IBM DataBlade Developers Kit User's Guide*.

Routine overloading, or polymorphism, refers to writing a routine that performs the same task and has the same name as an existing routine—but one that takes a different data type as an argument. When you create opaque, collection, or row

types, you can overload existing routines for your new data type. When the overloaded routine is called, your Informix database server determines which variant of the routine to use by examining the argument. BladeSmith creates a template for the overloaded routine; however, you must complete the code to enable the routine to complete the assigned task.

You can overload built-in and operator functions for all data types except built-in data types. You can create user-defined routines for all categories of data types. You can create support routines for all extended data types. You must create support routines for opaque data types (see "Opaque data type support routines" on page 1-15).

Built-in functions and operator functions

Built-in functions perform arithmetic and other basic operations when included in SQL statements. Operator functions are built-in functions that are bound to operator symbols.

For example, the **plus()** function is bound to the + operator. Some of the arithmetic functions take a single argument, such as the **root()** function, which calculates the square root of its argument; others take two arguments, such as the **pow()** function, which raises one argument to the power of the second.

When you overload a built-in or operator function, you must specify what the function does and what it returns. For example, if you overload the **plus()** function on a row type, you can choose to:

- add the respective field values and return a row type with the same number of fields as the input row types
- return a row type that contains all the fields of the first input row type followed by all the fields of the second input row type

There are also built-in functions that do not take arguments and that you cannot overload, such as the **today()** function, which returns the current date and time.

For more information, see the *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

User-defined routines

Typically, user-defined routines perform operations specific to the data or application for which they are created and are not based on routines provided with your IBM Informix database server.

Users call user-defined routines within SQL statements or through the DataBlade API. BladeSmith has a wizard to help you define the parameters for user-defined routines.

Casts

A cast is a rule that converts one data type into another.

Casts work in only one direction: from the source data type to the target data type. You can, however, define two casts for the same two data types to support conversion in both directions.

For some data types, you can choose whether the database server or the user controls casting.

Create an implicit cast if you want the database server to automatically convert the source data type.

Create an explicit cast if you want the user to specify the cast within an SQL statement.

If you are creating a cast between two data types that have different internal structures, you must write a cast support function. A straight cast, between two data types that have the same internal structure, does not require a cast support function; however, you can supply one to perform a conversion operation. You typically define straight casts to allow implicit casting from a distinct data type to its source data type (but not from a source data type to the distinct data type based on it).

You can use a built-in type as a source or target data type in a cast, but not as both. Built-in types have system-defined casts between each other that the database server invokes automatically.

A distinct type inherits all the casts of the source type. The database server automatically creates an explicit cast between the distinct type and the source type.

For more information about casting, see the *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Interfaces

An interface is a way to reference another DataBlade module within your DataBlade module.

Using an interface creates a dependency on the DataBlade module that provides the interface. You cannot register a DataBlade module that uses an interface unless the DataBlade module that provides the interface is already installed in the database server.

You can import an interface from another DataBlade module to facilitate development of your module. Similarly, you can build a DataBlade module that provides an interface for other DataBlade modules to use.

For more information, see the *IBM DataBlade Developers Kit User's Guide*.

Errors

You can define error or trace messages for your DataBlade module.

An error or trace consists of a unique five-character code, a locale (for translation), and a message. If you are developing a DataBlade module as a commercial product, qualify its name with a three-character DataBlade module prefix such as "USR" to create unique error codes and other DataBlade module objects.

To globalize your error messages, define multiple messages by using the same error code, a different locale, and the message text for that locale. Which message the user sees is controlled by the value of the locale environment variables.

For more information, see the *IBM DataBlade Developers Kit User's Guide*.

Unit tests

BladeSmith generates unit tests for your opaque data type support routines, user-defined routines, and cast support functions.

BladeSmith adds data to test boundary conditions for your data types. Use unit tests while you debug your DataBlade module by using Microsoft Visual C++ on Windows. Run unit tests with the DBDK Visual C++ Add-In and IfxQuery (see "DBDK Visual C++ Add-In and IfxQuery" on page 2-3).

For more information, see the *IBM DataBlade Developers Kit User's Guide*.

Functional tests

BladeSmith generates functional tests for your opaque data type support routines, user-defined routines, and cast support functions.

You must supply input data, the expected output data (if applicable), or an error code (if the input data is not valid) in BladeSmith. Run functional tests on UNIX after you finish coding your DataBlade module.

For more information, see the *IBM DataBlade Developers Kit User's Guide*.

Imported SQL files

You can include custom SQL statements in your DataBlade module to perform tasks such as creating and dropping user-defined access methods, support tables, or SPL routines.

You can include custom SQL statements by typing them in the SQL File wizard or by referencing a file.

You can specify dependencies between objects in your DataBlade module and your custom SQL. These dependencies determine the sequence in which the SQL statements are executed when you register the DataBlade module.

For more information, see the *IBM DataBlade Developers Kit User's Guide*.

Imported client files

After your customers install a DataBlade module on a database server, they might download any client files included in your DataBlade module to client computers.

The types of client files you can package with your DataBlade module include:

- graphical user interfaces
- documentation and help files
- shared object files, dynamic link libraries, or header files containing DataBlade module routines executed in the client address space.

Your customers use BladeManager to download the client files to their client computers.

For more information, see the *IBM DataBlade Developers Kit User's Guide*.

Chapter 2. Build a DataBlade module

DataBlade Developers Kit tools

The DataBlade Developers Kit includes graphical user interfaces for creating, packaging, and registering DataBlade modules, and tools for debugging DataBlade modules on Windows.

BladeSmith

You use BladeSmith to begin creating your DataBlade module, such as defining its contents and generating files and source code.

BladeSmith guides you through object definition with wizard pages. BladeSmith automates many of the tasks of object creation, such as writing the SQL statements necessary to define objects in the database.

Using BladeSmith, you create a project for your DataBlade module and then add or define the following types of objects for your module:

User-defined objects
Includes aggregates, casts, errors, interfaces, routines, and data types.

Files Can be custom SQL statements or files necessary for a client.

Imported objects
Includes built-in data types and interfaces from other DataBlade modules.

After you specify the user-defined objects, imported objects, and files you want to include in your DataBlade module, use BladeSmith to generate the files you need for compiling a shared object file or dynamic link library, managing a DataBlade module in your IBM Informix database server, testing object functionality, and creating installation packaging files. These categories of files are described in the following table.

Type of generated file	Description
Source code	You use these files to create a shared library file. They can include source code files, header files, Visual C++ workspace and project files, and makefiles.
SQL script	These files contain the SQL statements that support the DataBlade modules in the database system tables. They include prepare scripts that describe the DataBlade module and object scripts that describe the DataBlade objects.
Test	You use unit test files to test your database objects while you code and debug your DataBlade module on Windows. You use functional test files to test the positive and negative operation of user-defined routines, opaque data type support routines, and casts when your DataBlade module is complete.
Packaging	You use these files with BladePack to generate installation and executable files.

The generated source code files contain routine definitions. BladeSmith generates complete code for some routines, such as basic opaque data type support routines.

BladeSmith generates code templates for other types of routines, such as user-defined routines. You must add code to these routines to implement the functionality you require. The areas of the generated source code that need modification are marked with `TO DO:` comments.

When your code is complete, you compile it into a file that the database server can interpret.

For instructions, see the *IBM DataBlade Developers Kit User's Guide* or the tutorial in the information center.

BladePack

You use BladePack to create an installable DataBlade module package.

BladePack uses the packaging project file created by BladeSmith as the basis for the installation package. The packaging file references the SQL scripts, shared object file, and other files required by the DataBlade module. The installation scripts ensure that all DataBlade modules created with the DataBlade Developers Kit can be installed in a similar way.

You can create installation packages for a UNIX installation or a Windows installation for InstallShield 3.1 or InstallShield 5.1. The options you have for your installation package vary with each type of installation. For information about your options, see the *IBM DataBlade Developers Kit User's Guide*.

You can perform the following tasks with BladePack:

- Add files to your DataBlade module

  For example, you can include documentation, online help, and example files.
- Include several BladePack projects in an installation package

  For example, you can include DataBlade modules that facilitate similar financial calculations into a single installation package.
- Divide files into separate components, subcomponents, and shared components

  You can designate the components and subcomponents to include in typical and compact installations. You can also allow users to customize their installations by choosing the components and subcomponents they want to install. Shared components can belong to more than one component, and they are always installed with components to which they belong.
- Include custom installation routines

  You can add custom DLL routines, dialog boxes, and programs for Windows InstallShield 3.1 installations and custom programs for UNIX installations. You can also include readme files for any type of installation.
- Generate disk images or a directory tree for interactive installations

  On UNIX platforms, an interactive installation includes **install** and **uninstall** shell scripts. On Windows, an interactive InstallShield installation includes the **Setup** program and, for InstallShield 3.1, the **Uninstall** program.

For more information, see the *IBM DataBlade Developers Kit User's Guide*.

BladeManager

You use BladeManager to register or unregister your DataBlade module in a database and to install or uninstall DataBlade module client files.

After you install a DataBlade module on a database server, you must register it in every database that uses the module. Registration involves executing the DataBlade SQL scripts to create DataBlade objects in the database and making the DataBlade shared object or dynamic link library available to the database server.

BladeManager checks for dependencies between DataBlade modules. If you have imported an interface from another DataBlade module, BladeManager registers your DataBlade module only after it confirms that the interface is registered in the database.

If you are upgrading your DataBlade module, BladeManager automatically unregisters the previous version.

You can also unregister any module by using BladeManager. BladeManager does not allow you to unregister a DataBlade module if there is another DataBlade module that depends on it or if any of its objects are in use by the database.

For more information, see the *IBM Informix DataBlade Module Installation and Registration Guide*.

DBDK Visual C++ Add-In and IfxQuery

The IBM Informix DataBlade Developers Kit (DBDK) provides the DBDK Visual C++ Add-In and IfxQuery tools for debugging C and C++ code on Windows.

The DBDK Visual C++ Add-In is a toolbar you add to Microsoft Visual C++, Version 6.0, to aid in debugging. You must have Informix on the same computer as the DataBlade Developers Kit to use the debugging features of the add-in.

The IfxQuery tool is launched by the add-in from within Visual C++. IfxQuery runs the SQL unit test file in the active window in Visual C++.

The add-in and IfxQuery automate many of the steps necessary for debugging a DataBlade module. After you compile your DataBlade module in Visual C++ and set breakpoints in your source code, you can click **Debug DataBlade Module**.

The **Debug DataBlade Module** command performs the following steps:

1. Checks if the DataBlade module is compiled (and compiles it, if necessary)
2. If necessary, creates a directory for the DataBlade module under the `INFORMIXDIR\extend` directory
3. Installs the DataBlade module shared library file and SQL scripts in the `INFORMIXDIR\extend\`*`project`*`.0` directory
4. Shuts down the database server
5. Starts the Visual C++ debugger and the database server attached to the Visual C++ debugger
6. Launches IfxQuery, if the active window contains an SQL file
7. If necessary, creates the database specified by the add-in
8. Connects to the database specified by the add-in
9. Registers the DataBlade module
10. Executes the SQL statements from the unit test file
11. Writes the results to an HTML file
12. Launches the default HTML browser for your computer
13. Displays the SQL results in the HTML browser

The first time you run the **Debug DataBlade Module** command for a DataBlade module project, the Properties dialog box appears, in which you specify the database server and database you want to use for debugging. You can also access the Properties dialog box with the **Run Properties dialog box** button of the add-in.

How to create a DataBlade module

While the IBM Informix DataBlade Developers Kit (DBDK) tools run only on Windows, you can create DataBlade modules for both Windows and UNIX operating systems for the C, C++, and Java languages. The tools you use on each operating system and for each programming language vary.

The following table describes, in order, the tasks needed to create a DataBlade module and the tools you use to complete the tasks.

To perform this task	Use this tool on UNIX	Use this tool on Windows
Write the design and functional specifications.	Your word-processing program	Your word-processing program
Create your DataBlade module: • Set up a project for your DataBlade module. • Import objects from other DataBlade modules. • Define new objects for your DataBlade module. • Add validation test data for your new routines, opaque types, and casts. • Generate source code, SQL scripts, installation files, and unit and functional test files.		BladeSmith
Edit the source code to add C, C++, or Java code for routines, as needed.	Your development tool or text editor	For C or C++ code: Microsoft Visual C++ For Java code: your development tool or text editor
Compile your source code.	For C or C++ code: your compiler For Java code: JDK 1.1 compiler	For C or C++ code: Microsoft Visual C++ For Java code: JDK 1.1.*x* compiler
Install your DataBlade module.	Operating system copy command or FTP	For C or C++ code: DBDK Visual C++ Add-In For Java code: operating system copy of FTP
Register your DataBlade module in your test database.	For all code: • BladeManager • Informix[1]	For C or C++ code: • DBDK Visual C++ Add-In • Informix For Java code: • BladeManager • Informix[1]

To perform this task	Use this tool on UNIX	Use this tool on Windows
Debug your DataBlade module by running unit tests.	For C or C++ code: • a debugging utility • DB-Access or a client application • Informix For Java code: • the Java log file • DB-Access or a client application • Informix[1]	For C or C++ code: • Microsoft Visual C++ • DBDK Visual C++ Add-In • IfxQuery or other SQL query tool • Informix For Java code: • the Java log file • an SQL query tool • Informix[1]
Validate your DataBlade module with functional tests.	For all code: • DB-Access • Informix[1]	For all code: • MKS Toolkit • DB-Access • Informix[1]
Package your DataBlade module: • Add examples, online help files, and any other files you want to include to the project. • Define any additional components for a selective installation. • Perform optional customizations for installation packages. • Build the installation package.		BladePack
Transfer files to the installation media.	Your operating system	Your operating system
Document your DataBlade module with a users guide, release notes, examples, and online help, as needed.	Your word-processing program	Your word-processing program

Notes:

1. You must use the IBM Informix Dynamic Server with J/Foundation upgrade to IBM Informix to enable services that use Java. For more information about J/Foundation, see the *J/Foundation Developer's Guide*.

DataBlade module development resources

The DataBlade Developers Kit includes various resources to help you learn about and develop DataBlade modules.

The DataBlade Developers Kit Tutorial

The DataBlade Developers Kit Tutorial is a set of HTML documents that you access through the information center.

The tutorial consists of step-by-step exercises that demonstrate how to create DataBlade modules that extend your IBM Informix database server.

The first exercise demonstrates a simple DataBlade module so that you can focus on learning the mechanics of BladeSmith and the DBDK Visual C++ Add-In without complex coding. All tutorial users start with Exercise 1.

Each subsequent exercise is more complex than the previous one; you can choose to either work through the exercises sequentially or pick the ones that interest you.

Example DataBlade modules

Example DataBlade modules are included with the DataBlade Developers Kit software.

Example DataBlade modules are in the `%INFORMIXDIR%\dbdk\examples` directory. The information center has descriptions of the example DataBlade modules with links to readme files and source code.

The example DataBlade modules are frequently updated. The topics you might find covered in the example DataBlade modules include:

- **Client.** Using extended data types with ESQL/C and C++ client programs.
- **Routines.** Using user-defined routines written with the DataBlade API and using MMX technology.
- **Types.** Using extended data types that have support routines written with the DataBlade API, use MMX technology, are implemented as ActiveX value objects, and create user-defined statistics.

Appendix A. DataBlade module documentation

This section is a reference to current IBM Informix documentation pertaining to DataBlade modules.

This section is divided into three sections:

- Publication overview, a survey of the documentation set, arranged by concept
- Title-to-topic reference, a descriptive catalog of the documents, arranged alphabetically by title
- Topic-to-title reference, an alphabetical list of topics concerning DataBlade modules, with references to the document or documents that contain detailed information about each topic

For other DataBlade module resources, see "DataBlade module development resources" on page 2-5.

Publication overview

In the following table, the publications mentioned in the *IBM Informix DataBlade Module Development Overview* are arranged into basic categories.

Category	Publication Title
DataBlade module concepts	*IBM Informix User-Defined Routines and Data Types Developer's Guide* *IBM DataBlade Developers Kit User's Guide* *J/Foundation Developer's Guide*
DataBlade module development tools	*IBM DataBlade Developers Kit User's Guide* *IBM Informix DataBlade Module Installation and Registration Guide*
APIs	*IBM Informix DataBlade API Programmer's Guide* *IBM Informix DataBlade API Function Reference* *IBM Informix JDBC Driver Programmer's Guide* *IBM Informix GLS User's Guide* *IBM Informix ESQL/C Programmer's Manual*
IBM Informix	*IBM Informix Administrator's Guide* *IBM Informix Performance Guide*
SQL	*IBM Informix Guide to SQL: Reference* *IBM Informix Guide to SQL: Syntax* *IBM Informix Guide to SQL: Tutorial*

Title-to-topic reference

In the following table, the publications mentioned in the *IBM Informix DataBlade Module Development Overview* are listed alphabetically by title, with a brief description of each.

Publication title	Description
J/Foundation Developer's Guide	Describes how Java is implemented in the IBM Informix database server. Describes a library of classes and interfaces that allow programmers to create and execute Java user-defined routines that access Informix database servers.
IBM Informix DataBlade API Programmer's Guide *IBM Informix DataBlade API Function Reference*	Provide a complete reference for the DataBlade API, which is used to develop applications that interact with your Informix database server.
IBM DataBlade Developers Kit User's Guide	Describes how to develop and package DataBlade modules by using the DataBlade Developers Kit.
IBM Informix DataBlade Module Installation and Registration Guide	Explains how to install DataBlade modules and use BladeManager to register, upgrade, and unregister DataBlade modules in Informix databases.
IBM Informix User-Defined Routines and Data Types Developer's Guide	Explains how to extend existing data types, define new data types, and define your own functions and procedures for an Informix database. Describes the tasks you must perform to extend operations on data types, to create new casts, to extend operator classes for secondary access methods, and to write opaque data types. Defines common considerations for SPL and external routines and describes how to create user-defined aggregates.
IBM Informix R-Tree Index User's Guide	Describes the Informix R-tree secondary access method and how use its components. It describes how to create an R-tree index on appropriate data types and how to create an operator class that uses the R-tree access method to index a user-defined data type.
IBM Informix GLS User's Guide	Describes IBM Informix GLS, an application programming interface available in IBM Informix products. GLS provides ESQL/C and DataBlade module developers the ability to write programs (or change existing programs) to handle different languages, cultural conventions, and code sets.
IBM Informix ESQL/C Programmer's Manual	Explains how to use IBM Informix ESQL/C to create client applications with database management capabilities. This publication is a complete guide to the features of ESQL/C that allow you to gain access to Informix databases, manipulate the data in your program, interact with the database server, and check for errors.
IBM Informix Guide to SQL: Reference	Describes the Informix system catalog tables, common environment variables that you might need to set, and the built-in data types that your Informix database server supports.
IBM Informix Guide to SQL: Syntax	This publication contains syntax descriptions for the Structured Query Language (SQL) and Stored Procedure Language (SPL) statements that your Informix database server supports.
IBM Informix Guide to SQL: Tutorial	Includes instructions for using basic and advanced Structured Query Language (SQL), as well as for designing and managing your database.
IBM Informix Administrator's Guide	Describes how to install, configure, and use the features of your Informix database server.
IBM Informix Performance Guide	Explains how to configure and operate your database server to improve overall system performance as well as the performance of SQL queries.

Publication title	Description
IBM Informix JDBC Driver Programmer's Guide	Describes the JDBC driver that implements the Java interfaces and classes that programmers use to connect to an Informix database server.

Topic-to-title reference

The following table provides an alphabetical list of DataBlade module development topics, with references to the publications in which each topic is documented.

Topic	Detail/publication title
ActiveX value objects	*IBM DataBlade Developers Kit User's Guide*
Aggregates	*IBM Informix DataBlade API Programmer's Guide* *IBM DataBlade Developers Kit User's Guide*
APIs	DataBlade API: *IBM Informix DataBlade API Programmer's Guide* IBM Informix ESQL/C: *IBM Informix ESQL/C Programmer's Manual* IBM Informix GLS: *IBM Informix GLS User's Guide* IBM Informix JDBC Driver: *IBM Informix JDBC Driver Programmer's Guide*
BladeManager	*IBM Informix DataBlade Module Installation and Registration Guide* *IBM DataBlade Developers Kit User's Guide*
BladePack	*IBM DataBlade Developers Kit User's Guide*
BladeSmith	*IBM DataBlade Developers Kit User's Guide*
Casts	*IBM Informix User-Defined Routines and Data Types Developer's Guide* *IBM Informix Guide to SQL: Tutorial*
Coding standards	*IBM Informix DataBlade API Programmer's Guide* *J/Foundation Developer's Guide* *IBM DataBlade Developers Kit User's Guide*
Compiling source code	*IBM DataBlade Developers Kit User's Guide*
Data types	Using with user-defined routines: *IBM Informix DataBlade API Programmer's Guide* Built-in: *IBM Informix Guide to SQL: Reference* Qualified built-in: *IBM Informix Guide to SQL: Reference* Opaque: *IBM Informix User-Defined Routines and Data Types Developer's Guide* Distinct: *IBM Informix User-Defined Routines and Data Types Developer's Guide* Collection: *IBM Informix Guide to SQL: Tutorial* Row: *IBM Informix Guide to SQL: Tutorial*
DBDK Visual C++ Add-In	*IBM DataBlade Developers Kit User's Guide*
Dependencies between DataBlade modules	*IBM Informix DataBlade Module Installation and Registration Guide*
Error messages	*IBM DataBlade Developers Kit User's Guide* *IBM Informix DataBlade API Programmer's Guide*

Topic	Detail/publication title
Example DataBlade modules	Information center
Java value objects	*IBM DataBlade Developers Kit User's Guide*
Importing files	*IBM DataBlade Developers Kit User's Guide*
Inheritance	*IBM Informix Guide to SQL: Tutorial*
Installing DataBlade modules	*IBM Informix DataBlade Module Installation and Registration Guide* *IBM DataBlade Developers Kit User's Guide*
Interfaces to DataBlade modules	*IBM DataBlade Developers Kit User's Guide* DataBlade Developers Kit Tutorial
Memory management	*IBM Informix DataBlade API Programmer's Guide*
Operator class support functions	*IBM Informix User-Defined Routines and Data Types Developer's Guide* *IBM Informix R-Tree Index User's Guide*
Packaging a DataBlade module	*IBM DataBlade Developers Kit User's Guide*
Performance issues	*IBM Informix Performance Guide* *IBM Informix User-Defined Routines and Data Types Developer's Guide* *IBM Informix DataBlade API Programmer's Guide*
Polymorphism	*IBM Informix User-Defined Routines and Data Types Developer's Guide*
Registering a DataBlade module	*IBM Informix DataBlade Module Installation and Registration Guide*
Routines	*IBM Informix DataBlade API Programmer's Guide* *IBM Informix DataBlade API Function Reference* *IBM Informix User-Defined Routines and Data Types Developer's Guide* *J/Foundation Developer's Guide* *IBM DataBlade Developers Kit User's Guide*
Secondary access methods	*IBM Informix Performance Guide* *IBM Informix R-Tree Index User's Guide*
Server architecture	*IBM Informix Administrator's Guide*
Shared memory	*IBM DataBlade Developers Kit User's Guide* *IBM Informix Administrator's Guide*
Smart large objects	*IBM Informix DataBlade API Programmer's Guide* *IBM Informix Database Extensions User's Guide* *IBM Informix Guide to SQL: Tutorial*
SQL	*IBM Informix Guide to SQL: Reference* *IBM Informix Guide to SQL: Syntax* *IBM Informix Guide to SQL: Tutorial*
Storage of DataBlade modules	*IBM Informix Administrator's Guide*
Stored Procedure Language	*IBM Informix User-Defined Routines and Data Types Developer's Guide* *IBM Informix Guide to SQL: Syntax*

Topic	Detail/publication title
Testing and debugging DataBlade modules	*IBM DataBlade Developers Kit User's Guide*
Unit tests	*IBM DataBlade Developers Kit User's Guide*
Virtual processors	*IBM DataBlade Developers Kit User's Guide* *IBM Informix Administrator's Guide*

Appendix B. Informix DataBlade modules

IBM Informix DataBlade modules serve as examples of the type of functionality DataBlade modules can provide.

IBM Informix Excalibur Text Search DataBlade Module

The IBM Informix Excalibur Text Search DataBladeModule enables you to search your data in ways that are faster and more sophisticated than the keyword matching that SQL provides.

Excalibur text search capabilities include phrase matching, exact and fuzzy searches, compensation for misspelling, and synonym matching. The Informix Excalibur Text Search DataBlade Module can search any type of text.

The Informix Excalibur Text Search DataBlade Module uses dynamic links in the Excalibur class library, or text search engine, to perform the text search section of the SELECT statement instead of having the database server perform a traditional search. The text search engine is designed to perform sophisticated and fast text searches. It runs in one of the database server-controlled virtual processes.

Extensions to Informix

The IBM Informix Excalibur Text Search DataBladeModule provides four kinds of objects to extend your Informix database server: the **etx** access method, the filter utility, the **etx_contains()** operator, and text search routines.

The **etx** access method allows you to call on the Excalibur Text Retrieval Library to create indexes that support sophisticated searches on table columns that contain text. The indexes that you create with the **etx** access method are called etx indexes.

To take advantage of the **etx** access method, you must store the data you want to search—called search text—in a column of type IfxDocDesc, BLOB, CLOB, CHAR, VARCHAR, or LVARCHAR. The first data type in this list, IfxDocDesc, is a data type designed specifically for use with text access methods. The most popular data types for large documents are BLOB and CLOB.

When you store your documents in a column, you do not need to convert them from their proprietary format into ASCII when creating an **etx** index; the Informix Excalibur Text Search DataBlade Module does this for you. One of the components of the Informix Excalibur Text Search DataBlade Module is a filtering utility that recognizes a number of document formats and converts them into ASCII form whenever needed.

You use the **etx_contains()** operator within SELECT statements to perform searches of **etx** indexes.

In addition to the **etx_contains()** operator, the Informix Excalibur Text Search DataBlade Module supplies several routines that you can use to perform tasks such as creating and dropping synonym and stopword lists.

For more information, see the *Excalibur Text Search DataBlade Module User's Guide*.

IBM Informix Web DataBlade Module

The IBM Informix Web DataBlade Module enables you to create web applications that incorporate data retrieved dynamically from an IBM Informix database.

Using the Informix Web DataBlade module, you do not need to develop a Common Gateway Interface (CGI) application to dynamically access database data. Instead, you create HTML pages that include Informix Web DataBlade module tags and functions that dynamically execute the SQL statements you specify and format the results. These pages are called application pages (AppPages). The types of data you retrieve can include traditional data types, HTML, image, audio, and video data.

Extensions to Informix

The IBM Informix Web DataBlade Module consists of three main components:

Webdriver
As a client application to your IBM Informix database server, Webdriver builds the SQL queries that execute the **WebExplode** function to retrieve AppPages from the database. Webdriver returns the HTML resulting from calls to the **WebExplode** function to the web server.

WebExplode function
The **WebExplode** function builds dynamic HTML pages based on data stored in the database. **WebExplode** parses AppPages that contain Informix Web DataBlade module tags within HTML and dynamically builds and executes the SQL statements and processing instructions embedded in the Informix Web DataBlade module tags. **WebExplode** formats the results of these SQL statements and processing instructions and returns the resulting HTML page to the client application (usually Webdriver). The SQL statements and processing instructions are specified by using SGML-compliant processing tags.

Informix Web DataBlade module tags and attributes
The Informix Web DataBlade module includes its own built-in set of SGML-compliant tags and attributes that enable SQL statements to be executed dynamically within AppPages.

The following diagram illustrates the architecture of the Informix Web DataBlade module.

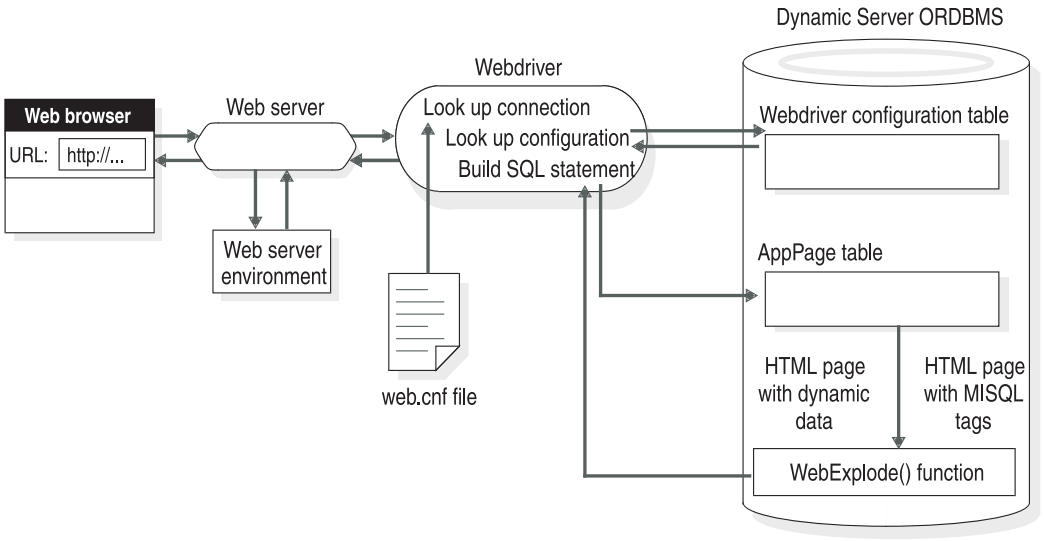

Figure B-1. Informix Web DataBlade module architecture

When a web address contains a Webdriver request, the web browser makes a request to the web server to start Webdriver. Based on configuration information from both a file on the operating system file system (`web.cnf`) and Webdriver configuration information stored in a database, Webdriver composes an SQL statement to retrieve the requested AppPage and then executes the **WebExplode** function. **WebExplode** retrieves the requested AppPage from the web application table (stored in the database), executes the SQL statements within that AppPage by expanding the Informix Web DataBlade module tags, and formats the results. **WebExplode** returns the resulting HTML to Webdriver. Webdriver returns the HTML to the web server, which returns the HTML to be rendered by the web browser.

Webdriver also enables you to retrieve large objects, such as images, directly from the database when you specify a path that identifies a large object stored in the database.

For more information, see the *IBM Informix Web DataBlade Module Application Developer's Guide*.

Appendix C. Accessibility

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

Accessibility features for IBM Informix products

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 products. These features support:

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

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.

IBM and accessibility

For more information about the IBM commitment to accessibility, see the *IBM Accessibility Center* at http://www.ibm.com/able.

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, that element is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 refers 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 can write `HOST STATE`, but you cannot 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 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

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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites 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 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, 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 is for planning purposes only. The information herein is subject to change before the products described become available.

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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

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.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Index

A

Accessibility C-1
  dotted decimal format of syntax diagrams C-1
  keyboard C-1
  shortcut keys C-1
  syntax diagrams, reading in a screen reader C-1
ActiveX value objects 1-6, 1-8
Aggregates
  defined 1-9
  where documented A-3
API. 1-5
Application programming interface
  DataBlade API 1-5
  JDBC 1-6
  supplied in Client SDK 1-8
  where documented A-3
Applications, simplifying with DataBlade modules 1-2

B

BladeManager
  defined 2-3
  registering DataBlade modules with 2-4
BladePack
  DataBlade module packaging tasks 2-4
  defined 2-2
  packaging options 2-2
BladeSmith
  aggregates, creating with 1-9
  casts, creating with 1-16
  client files, importing with 1-18
  collection data types
    creating with 1-11
  data type categories in 1-10
  DataBlade module creation tasks 2-4
  defined 2-1
  distinct data types
    creating with 1-11
  errors, creating with 1-17
  functional tests, generating with 1-18
  generated files from 2-1
  interfaces
    creating with 1-17
  opaque data type support routines
    creating with 1-15
  opaque data types, creating with 1-13
  overloading routines with 1-15
  qualified data types, specifying with 1-10
  routines
    creating with 1-15
  row data types, creating with 1-12
  SQL files, importing with 1-18
  unit tests, generating with 1-18
Built-in data types 1-10
Built-in routines 1-16

C

C code in DataBlade modules 1-5
C++ code in DataBlade modules 1-6
Casts
  defined 1-16
  where documented A-3
Client files, adding with BladeSmith 1-18
Client objects
  using with DataBlade modules 1-8
Client Software Development Kit 1-8
Client visualization tools 1-8
Codes for errors 1-17
Coding standards, documentation of A-3
Collection data types
  defined 1-11
  elements of 1-11
  illustration of 1-11
  LIST type constructor for 1-11
  MULTISET type constructor for 1-11
  return values, using as 1-11
  SET type constructor for 1-11
  where documented A-3
Combining DataBlade modules 1-2
Compiling source code
  tools for 2-4
  where documented A-3
compliance with standards vi
Constructors. 1-11
Converting data types 1-16
Creating DataBlade modules 2-4

D

Data types
  built-in 1-10
  categories of 1-10
  collection 1-11
  converting 1-16
  defined 1-10
  distinct 1-11
  documentation of A-3
  opaque 1-13
  qualified built-in 1-10
  row 1-12
DataBlade API 1-5
DataBlade Developers Kit
  example DataBlade modules 2-6
  overview 1-3
  tools in 2-1
  tutorial 2-5
DataBlade modules
  advantages 1-3
  aggregates in 1-9
  C code for 1-5
  C++ code for 1-6
  casts in 1-16
  client files with 1-18
  client objects in 1-8
  combining 1-2
  compiling source code for 2-4
  components 1-8
  components of 1-18
  creation task list for 2-4
  data types in 1-10, 1-12

DataBlade modules *(continued)*
debugging 2-4
defined 1-1
dependencies between A-3
development resources for 2-5
documenting 2-4
editing source code for 2-4
errors in 1-17
examples of A-3
extending IBM Informix with 1-1
foundation, using as 1-3
functional tests for 1-18
IBM Informix B-1, B-2
imported files in 1-18
installation files for 2-1
installing 2-4, A-3
interfaces in 1-3, 1-17
Java code for 1-6
location of examples of 2-6
memory allocation for 1-7
programming languages for 1-5
registering in a database 2-3
routines in 1-15
server architecture, role within 1-4
simplifying applications with 1-2
source code for 2-1
SQL scripts for 2-1
storage for A-3
Stored Procedure Languages, in 1-6
test files 2-1
testing 2-4, A-3
transaction control for 1-2
tutorial for 2-5
unit tests for 1-18
virtual processors, using with 1-7
DBDK Visual C++ Add-In
debugging a DataBlade module with 2-4
defined 2-3
documentation of A-3
registering DataBlade modules with 2-4
Debug DataBlade Module command 2-3
Debugging DataBlade modules
on UNIX 2-4
on Windows 2-3, 2-4
Development resources for creating DataBlade modules 2-5
Disabilities, visual
reading syntax diagrams C-1
Disability C-1
Distinct data types 1-11
Documentation
list of A-1, A-3
Tutorial 2-5
Documenting a DataBlade module 2-4
Dotted decimal format of syntax diagrams C-1

E

Editing source code 2-4
Elements, of collections 1-11
Errors
defined 1-17
where documented A-3
Example DataBlade modules 2-6
Excalibur Text Search DataBlade module B-1
Explicit casts 1-16

F

Fields
row data types 1-12
Files generated by BladeSmith 2-1
Files imported into DataBlade modules 1-18
Foundation DataBlade modules
defined 1-3
Functional indexes 1-1
Functional tests for DataBlade modules 1-18
Functions 1-15

G

Generated files from BladeSmith 2-1

I

IBM Informix
advantages of extending 1-1
architecture of 1-4, A-3
components 1-4
extended by DataBlade modules 1-1
improving performance 1-1
memory allocation for DataBlade modules in 1-7
transaction control for DataBlade modules in 1-2
using Client SDK with 1-8
virtual processors in 1-7
IBM Informix Web DataBlade module B-2
IfxQuery 2-3
Illustrations
collection data type 1-11
IBM Informix architecture 1-4
opaque data type 1-13
row data type 1-12
Implicit cast 1-16
Imported files
defined 1-18
where documented A-3
Indexes, functional 1-1
industry standards vi
Inheritance
in row data types 1-12
where documented A-3
Installation files for DataBlade modules 2-1
Installation packaging options 2-2
Installing a DataBlade module
tools for 2-4
where documented A-3
Interfaces
defined 1-3
importing 1-17
where documented A-3

J

Java code in DataBlade modules 1-6
Java value objects
about 1-6, 1-8
where documented A-3
JDBC API 1-6

L

Large objects. A-3
LIST type constructor 1-11

Locales for errors 1-17

M

Memory allocation
for DataBlade modules 1-7
where documented A-3
MULTISET type constructor 1-11

O

Opaque data types
ActiveX value objects, implemented as 1-6
defined 1-13
illustrated 1-13
Java value objects, implementing as 1-6
rich media data, using for 1-13
smart large objects in 1-13
support functions for 1-15
where documented A-3
Operator class support functions A-3
Operator functions 1-16
Overloading routines
defined 1-15
where documented A-3

P

Packaging DataBlade modules
options for 2-2
using BladePack for 2-4
where documented A-3
Performance
improving with DataBlade modules 1-1
where documented A-3
Polymorphism 1-15
Procedures 1-15
Programming languages for DataBlade modules 1-5

Q

Qualified built-in data types 1-10

R

Registering DataBlade modules
defined 2-3
where documented A-3
with BladeManager 2-4
with the DBDK Visual C++ Add-In 2-4
Routines
adding code to 2-1
built-in 1-16
categories of 1-15
defined 1-15
opaque type support 1-15
operator 1-16
overloading 1-15
returning collections with 1-11
user-defined 1-16
where documented A-3
Row data types
defined 1-12
fields in 1-12
illustration of 1-12
Row data types *(continued)*
inheritance for 1-12
smart large objects in 1-12

S

Screen reader
reading syntax diagrams C-1
Secondary access methods
defined 1-1
example of B-1
where documented A-3
SET type constructor 1-11
Shared memory
defined 1-7
where documented A-3
Shortcut keys
keyboard C-1
Smart large objects
in opaque data types 1-13
in row data types 1-12
where documented A-3
Source code
compiling 2-4
editing 2-4
files for DataBlade modules 2-1
SPL. 1-6
SQL
documentation of A-3
files included in DataBlade modules 1-18
scripts for DataBlade modules 2-1
standards vi
Storage for DataBlade modules A-3
Stored Procedure Languages
using in DataBlade modules 1-6
where documented A-3
Straight casts 1-16
Support functions
for casts 1-16
for opaque data types 1-15
Syntax diagrams
reading in a screen reader C-1

T

Test files for DataBlade modules 2-1
Testing a DataBlade module
tools for 2-4
Unit tests
where documented A-3
Virtual processors
where documented A-3
where documented A-3
Text Search DataBlade module B-1
Tutorial 2-5
Type constructors
LIST 1-11
MULTISET 1-11
SET 1-11

U

Unit tests
defined 1-18
executing with IfxQuery 2-3
using during debugging 2-4

Unregistering a DataBlade module 2-3
Upgrading a DataBlade module 2-3
User-defined routines 1-16
User-defined virtual processors 1-7

V

Virtual processors
  defined 1-7
Visual disabilities
  reading syntax diagrams C-1

W

Web DataBlade module B-2

Printed in USA

GC27-5555-00