DataBlade Module Development Overview

Version 4.0A
March 2003
Part No. CT1V0NA

Note:
Before using this information and the product it supports, read the information in the appendix entitled “Notices.”

Table of Contents

Introduction

In This Introduction

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

About This Manual

This manual is an overview of the DataBlade module development process. A DataBlade module extends the functionality of IBM Informix Dynamic Server 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.

This section discusses the organization of the manual, the intended audience, and the associated software products you must have to develop and use a DataBlade module.

Organization of This Manual

This manual begins with a conceptual overview of DataBlade modules and then provides a procedural overview of the development of a DataBlade module. Two appendixes describe current DataBlade module documentation and DataBlade modules available from IBM.

This manual provides a basic orientation for DataBlade module development. See Appendix A, "DataBlade Module Documentation," for a reference guide to current Informix documentation on DataBlade modules and related topics.

This manual includes the following chapters:

- Chapter 1, "DataBlade Module Concepts," introduces DataBlade modules. It includes the definition of a DataBlade module, an overview of how DataBlade modules fit into Dynamic Server architecture, descriptions of DataBlade module components, and DataBlade module programming language options.
- Chapter 2, "Building a DataBlade Module," provides a description of the DataBlade Developers Kit tools, a list of DataBlade module development tasks, and a description of programming resources and examples.
- Appendix A, "DataBlade Module Documentation," is a reference guide to current IBM Informix documentation on DataBlade modules. The appendix is divided into three sections:
  - An overview of the documentation set, arranged by concept
  - A descriptive catalog of the documents, arranged alphabetically by title
  - An alphabetical list of topics concerning DataBlade modules, with references to the document or documents that contain detailed information about each topic
- Appendix B, "IBM Informix DataBlade Modules," introduces several IBM Informix DataBlade modules and describes how they extend Dynamic Server.
- A Notices appendix describes IBM products, features, and services.
- A glossary of relevant terms follows the chapters, and an index directs you to areas of particular interest.

Types of Users

This guide is 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 Informix DataBlade Developer's Kit User's Guide* provides technical information specifically for developers who are ready to develop DataBlade modules.

Software Dependencies

See the *Read Me First* sheet for the DataBlade Developers Kit for complete system requirements for DBDK.

Typographical Conventions

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

Convention	Meaning
KEYWORD	All primary elements in a programming language statement (keywords) appear in uppercase letters in a serif font.
italics ***italics*** *`italics`*	Within text, new terms and emphasized words appear in italics. Within syntax and code examples, variable values that you are to specify appear in italics.
boldface ***boldface***	Names of program entities (such as classes, events, and tables), environment variables, file and pathnames, and interface elements (such as icons, menu items, and buttons) appear in boldface.
`monospace` *`monospace`*	Information that the product displays and information that you enter appear in a monospace typeface.
♦	This symbol indicates the end of product- or platform-specific information.
→	This symbol indicates a menu item. For example, "Choose **Tools→Options**" means choose the **Options** item from the **Tools** menu.

Tip: *When you are instructed to "enter" characters or to "execute" a command, immediately press* RETURN *after the entry. When you are instructed to "type" the text or to "press" other keys, no* RETURN *is required.*

Additional Documentation

This section lists resources that contain detailed information about the topics introduced in this manual.

Related Reading

For a complete reference to DataBlade module documentation, see Appendix A, "DataBlade Module Documentation."

The IBM Informix Developer Zone

The IBM Informix Developer Zone contains numerous white papers, code examples, and tips for creating DataBlade modules. You can find it at www.ibm.com/software/data/developer/informix.

Online Documentation

The DataBlade Developers Kit includes online help, the DataBlade Developers Kit InfoShelf, online manuals, release notes, and documentation notes.

Online Help

The DataBlade Developers Kit tools include context-sensitive help that provides both overview and detailed reference information for these tools.

The DataBlade Developers Kit InfoShelf

The DataBlade Developers Kit InfoShelf is a set of HTML documents to help you develop DataBlade modules. The InfoShelf contains a tutorial, descriptions of example DataBlade modules, and full-text IBM Informix manuals.

See "The DataBlade Developers Kit InfoShelf" on page 2-11 for more information on the InfoShelf.

Online Manuals

All the DataBlade Developers Kit manuals are also provided on the IBM Informix Online Documentation site at http://www.ibm.com/software/data/informix/pubs/library/.

Release Notes and Documentation Notes

The following on-line release notes and documentation notes, which you can access by clicking their icons in the **Informix** program group, supplement the information provided in this manual.

Online File	Purpose
dbdkdoc.txt	Describes features that are not covered in the manuals or that have been modified since publication.
dbdkrel.txt	Describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.

These files are located in the **%INFORMIXDIR%\release** directory under the subdirectory for your locale. Please examine these files because they contain vital information about application and performance issues.

IBM Welcomes Your Comments

To help us with future versions of our manuals, let us know about any corrections or clarifications that you would find useful. Include the following information:

- The name and version of your manual
- Any comments that you have about the manual
- Your name, address, and phone number

Send electronic mail to us at the following address:

docinf@us.ibm.com

This address is reserved for reporting errors and omissions in our documentation. For immediate help with a technical problem, contact Customer Services.

Chapter

DataBlade Module Concepts

1

In This Chapter

This chapter introduces DataBlade modules in the following sections:

- "What Are DataBlade Modules?" next
- "DataBlade Modules and the Database Server" on page 1-9
- "DataBlade Module Components" on page 1-15
- "DataBlade Module Programming Languages" on page 1-11

What Are DataBlade Modules?

A *DataBlade module* is a software package that extends the functionality of IBM Informix Dynamic Server. 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 Dynamic Server over using traditional relational databases and applications are:

- better performance.
- simpler applications.
- transaction control.
- scalability.

Better Performance

Your 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 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 Dynamic Server scale to a large number of users just as well as the database server itself.

Why Create a DataBlade Module?

You can extend your 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 include:

- control.
- 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 specifically 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.** To create your DataBlade module.
- **BladePack.** To package your DataBlade module.
- **BladeManager.** To make 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.** To debug your DataBlade module within Microsoft Visual C++.
- **IfxQuery.** To execute 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 Informix database server and ensures consistency among your user-defined routines.

DataBlade Modules and the Database Server

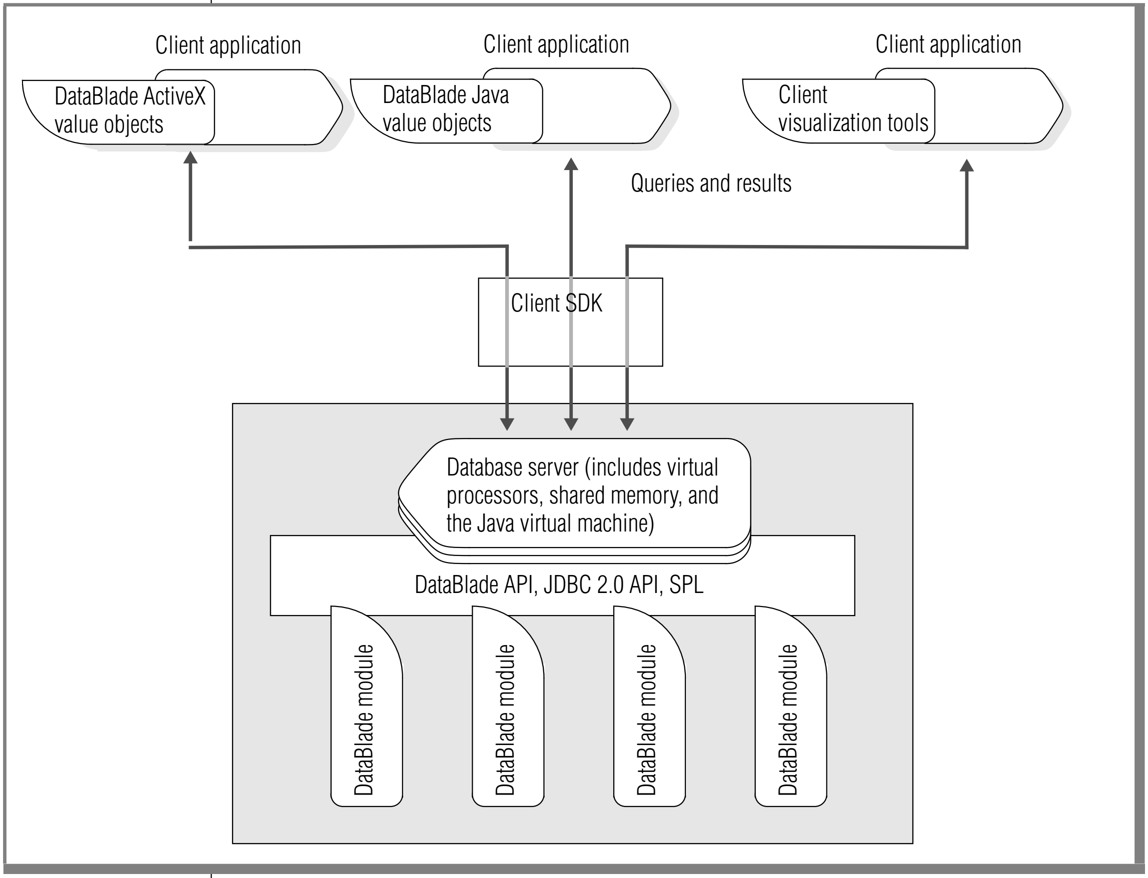
This section discusses the overall architecture of the Informix database server, how DataBlade modules affect database server processes, and the application programming interfaces you can use in your DataBlade modules and client applications.

Figure 1-1 on page 1-10 illustrates the following components of 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 Client Software Developer's 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 Dynamic Server with DataBlade Modules


Client application
DataBlade ActiveX value objects
Client application
DataBlade Java value objects
Client application
Client visualization tools
Queries and results
Client SDK
Database server (includes virtual processors, shared memory, and the Java virtual machine)
DataBlade API, JDBC 2.0 API, SPL
DataBlade module
DataBlade module
DataBlade module
DataBlade module

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

Important: *You must use the IBM Informix Dynamic Server with J/Foundation upgrade to IBM Informix Dynamic Server to enable services that use Java. For more information about J/Foundation, see the manual "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 on programming language options and restrictions, see the *IBM Informix DataBlade Developer's Kit User's Guide.*

C Language

The DataBlade Developers Kit 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 DataBlade Developers Kit 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 Dynamic Server on Windows only. Check the IBM Informix Developer Zone at www.ibm.com/software/data/developer/informix for the latest recommendations on C++ programming options.*

Java Language

The DataBlade Developers Kit 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 on IBM Informix JDBC Driver, see the *IBM Informix JDBC Driver Programmer's Guide*.

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

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

Informix Stored Procedure Language

You can use Informix Stored Procedure Language (SPL) statements to write routines, and you can store these SPL routines in the database. 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 on 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 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 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 routines execute in the CPU VP; however, if your DataBlade module routine makes use of 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 Dynamic Server Administrator's Guide* and the IBM Informix Developer Zone at www.ibm.com/software/data/developer/informix.

DataBlade Module Memory Allocation

Another important aspect of the internal architecture of your 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* manual for complete information.

Java Virtual Machine

Dynamic Server 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 mainly 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 on how the database server handles Java code, see the *J/Foundation Developer's Guide* manual.

The Client Software Developer's Kit

The Client Software Developer's Kit (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
- 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 *IBM Informix Client Products Installation Guide for Microsoft Windows Environments* or *IBM Informix Client Products Installation Guide for UNIX*.

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 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 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.** To perform user-defined computations on data.
- **Data Types.** To characterize data to the database server (either built-in data types or new data types).
- **Routines.** To operate on or return data.
- **Casts.** To convert data from one type to another.
- **Interfaces.** To create dependencies between DataBlade modules.

- **Errors.** To create messages raised by user-defined routines that appear as standard database server messages.
- **Unit Tests**. To test your database objects during the coding and debugging cycle.
- **Functional Tests.** To validate your completed DataBlade module routines.
- **Imported SQL Files.** To include custom SQL statements to create tables, user-defined access methods, and other database objects in your DataBlade module.
- **Imported Client Files.** To 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. The objects are described in the following sections.

Important: *Not all database objects and options described in this section are available with all versions of Dynamic Server. For more information, see the "IBM Informix DataBlade Developer's 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 on these aggregates, see the *IBM Informix Guide to SQL: Syntax*.

For more information on defining aggregates, see the *IBM Informix DataBlade Developer's 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 Informix data type that comes with the database server
Qualified built-in	Yes	No	A built-in data type that takes one or more qualifications, such as storage size, range of values, or precision
Opaque	Yes	Yes (the basic code needed to implement an opaque type is generated by BladeSmith)	A structured data type whose internal members cannot be accessed directly using SQL statements
Distinct	Yes	No	A unique 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 your Informix database server. 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* manual.

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. 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 to the right of the decimal point).

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

Example

The DECIMAL(6,3) data type has six digits, with three digits to the right of the decimal point; for example, 123.456.

More Information

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

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.

Why Use a Distinct Data Type?

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 very 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.

More Information

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 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.

What Does a Collection Look Like?

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

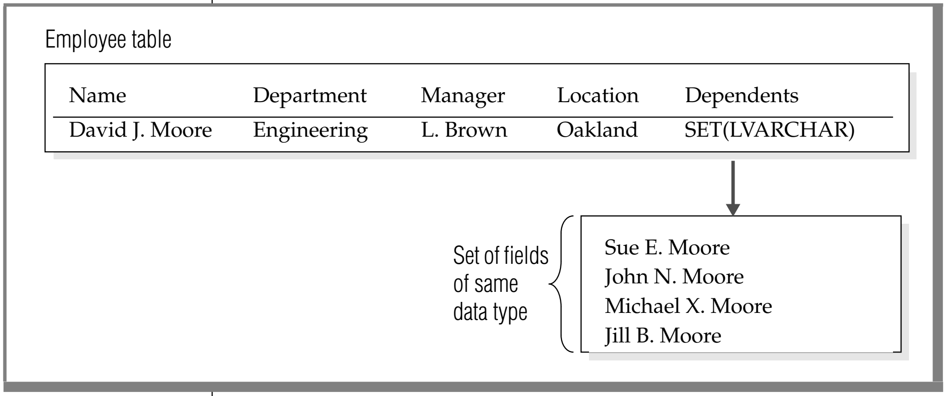


Figure 1-2
Sample Collection Data Type

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

Why Use a Collection Data Type?

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

More Information

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

Row Data Types

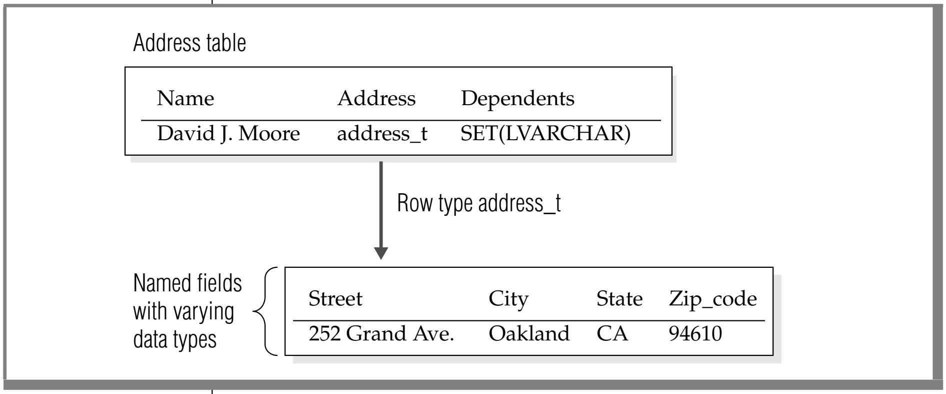
A row data type can be thought of as a row of columns, of varying data types, stored in a single database table column. Row data types follow essentially the same rules as database tables. The columns within a row data type are called *fields*. They can be almost any data type, including other extended data types and built-in data types, such as smart large objects. You can access fields individually 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.

What Does a Row Data Type Look Like?

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



Name	Address	Dependents
David J. Moore	address_t	SET(LVARCHAR)

Street	City	State	Zip_code
252 Grand Ave.	Oakland	CA	94610

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).

Why Use a Row Data Type?

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 row data type's fields.

You can use row data types to create inheritance hierarchies, allowing you to write very selective routines. A child row data type inherits its parent's fields 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.

More Information

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

Opaque Data Types

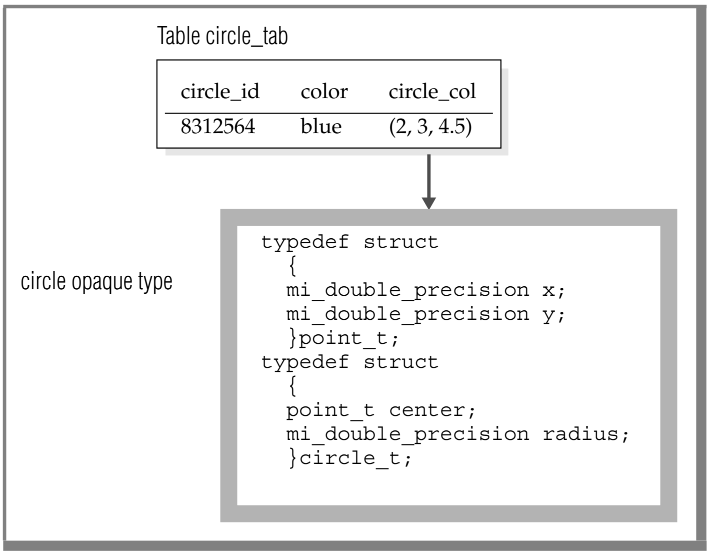
An *opaque* data type is a user-defined data structure. To successfully interpret opaque data types, the database server requires that the DataBlade module provide opaque data type support routines. You must provide support routines for your opaque data type.

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 1-25 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.

What Does an Opaque Data Type Look Like?

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



circle_id	color	circle_col
8312564	blue	(2, 3, 4.5)

```
typedef struct
  {
  mi_double_precision x;
  mi_double_precision y;
  }point_t;
typedef struct
  {
  point_t center;
  mi_double_precision radius;
  }circle_t;
```

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.

Why Use an Opaque Data Type?

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.*

(1 of 2)

Function	Recommended for	Description
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 multirepresentational 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.*

(2 of 2)

More Information

For a description of creating opaque types and their support routines, see the *IBM Informix DataBlade Developer's Kit User's Guide* or *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 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 Informix DataBlade Developer's 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 1-25).

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 Informix database server. End 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 Informix DataBlade Developer's 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 localize your error messages, define multiple messages 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 Informix DataBlade Developer's 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 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 2-7).

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

Imported Client Files

After your customers install a DataBlade module on a database server, they may 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 Informix DataBlade Developer's Kit User's Guide*.

Chapter

2

Building a DataBlade Module

In This Chapter

This chapter introduces topics discussed in the following sections:

- "DataBlade Developers Kit Tools," next
- "How to Create a DataBlade Module" on page 2-8
- "DataBlade Module Development Resources" on page 2-11

DataBlade Developers Kit Tools

The following graphical user interfaces are provided for creating, packaging, and registering DataBlade modules, as well as tools for debugging DataBlade modules on Windows:

- BladeSmith
- BladePack
- BladeManager
- DBDK Visual C++ Add-In and IfxQuery

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.

Each of these objects is summarized in Chapter 1, "DataBlade Module Concepts."

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 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 files and executables.

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 Informix DataBlade Developer's Kit User's Guide* or the tutorial in the on-line DataBlade Developers Kit InfoShelf.

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 on your options, see the *IBM Informix DataBlade Developer's Kit User's Guide*.

You can perform the following tasks with BladePack:

- Add files to your DataBlade module

  For example, you can include documentation, on-line 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.

See the *IBM Informix DataBlade Developer's Kit User's Guide* for more information.

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.

See the *IBM Informix DataBlade Module Installation and Registration Guide* for more information.

DBDK Visual C++ Add-In and IfxQuery

The DataBlade Developers Kit provides the following tools for debugging C and C++ code on Windows:

- DBDK Visual C++ Add-In
- IfxQuery

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 Dynamic Server 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 new 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 DataBlade Developer Kit 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 should 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

(1 of 3)

To Perform This Task	Use This Tool on UNIX	Use This Tool on Windows
Register your DataBlade module in your test database.	For all code: ■ BladeManager ■ Dynamic Server*	For C or C++ code: ■ DBDK Visual C++ Add-In ■ Dynamic Server For Java code: ■ BladeManager ■ Dynamic Server*
Debug your DataBlade module by running unit tests.	For C or C++ code: ■ a debugging utility ■ DB-Access or a client application ■ Dynamic Server For Java code: ■ the Java log file ■ DB-Access or a client application ■ Dynamic Server*	For C or C++ code: ■ Microsoft Visual C++ ■ DBDK Visual C++ Add-In ■ IfxQuery or other SQL query tool ■ Dynamic Server For Java code: ■ the Java log file ■ an SQL query tool ■ Dynamic Server*
Validate your DataBlade module with functional tests.	For all code: ■ DB-Access ■ Dynamic Server*	For all code: ■ MKS Toolkit ■ DB-Access ■ Dynamic Server*
Package your DataBlade module: ■ Add examples, on-line 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

(2 of 3)

To Perform This Task	Use This Tool on UNIX	Use This Tool on Windows
Transfer files to the installation media.	Your operating system	Your operating system
Document your DataBlade module with a user's guide, release notes, examples, and online help, as needed.	Your word processing program	Your word processing program

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

(3 of 3)

DataBlade Module Development Resources

The DataBlade Developers Kit includes various resources to help you learn about and develop DataBlade modules, discussed in the following subsections:

- "The DataBlade Developers Kit InfoShelf," next
- "The Tutorial" on page 2-12
- "Example DataBlade Modules" on page 2-13
- "The IBM Informix Developer Zone" on page 2-14

The DataBlade Developers Kit InfoShelf

The DataBlade Developers Kit InfoShelf is a set of HTML documents included with the DataBlade Developers Kit software.

The DataBlade Developers Kit InfoShelf can be launched from the BladeSmith **Help** menu or started independently from the **Informix** program group.

The InfoShelf provides the following information in HTML format:

- An online version of this manual
- A tutorial that illustrates the fundamentals of DataBlade module development

- Descriptions of example DataBlade modules shipped with the DataBlade Developers Kit
- A reference library that contains the following manuals:
  - *IBM Informix DataBlade Developer's Kit User's Guide*
  - *IBM Informix DataBlade Module Installation and Registration Guide*
  - *IBM Informix DataBlade API Programmer's Guide*
  - *IBM Informix User-Defined Routines and Data Types Developer's Guide*
  - *IBM Informix Guide to SQL: Reference*
  - *IBM Informix Guide to SQL: Syntax*
  - *IBM Informix Guide to SQL: Tutorial*
  - *J/Foundation Developer's Guide*
  - *IBM Informix JDBC Driver Programmer's Guide*
  - *IBM Informix GLS Programmer's Manual*
  - *IBM Informix GLS User's Guide*
  - *IBM Informix ESQL/C Programmer's Manual*
- A master index containing the merged index entries of all the books listed above

  The index entries provide links into the HTML versions of the manuals included in the InfoShelf.

The Tutorial

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

The tutorial consists of step-by-step exercises that demonstrate how to create DataBlade modules that extend your 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 should 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 just 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 InfoShelf 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.

The IBM Informix Developer Zone

The Informix Developer Zone Web site is designed to provide you with the technical information, tools, forums, and links to relevant information that you need when using Informix products.

The DataBlade Developers Kit InfoShelf contains links to the IBM Informix Developer Zone at www.ibm.com/software/data/developer/informix. This page has a link to the DataBlade Developers' Corner, which is of particular interest to DataBlade module developers. Its goal is to answer questions asked by DataBlade developers. The information in the DataBlade Developers' Corner is frequently updated, and it includes topics such as:

- **Getting started.** Lists software, documentation, and training resources and describes how to install and set up the requisite software.
- **DataBlade modules and your Informix database server.** Includes descriptions of the interaction between DataBlade modules and your database server and how to debug DataBlade modules.
- **DataBlade API.** Includes tips and code examples, from snippets to complete DataBlade modules, using the DataBlade API.
- **Data types.** Describes the data types you can use with your Informix database server, such as extended data types and smart large objects.

DataBlade Module Documentation

This appendix is a reference to current IBM Informix documentation pertaining to DataBlade modules. The appendix is divided into three sections:

- "Manual 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 2-11.

Manual Overview

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

Category	Manual Title
DataBlade module concepts	*IBM Informix User-Defined Routines and Data Types Developer's Guide*
	IBM Informix DataBlade Developer's Kit User's Guide
	J/Foundation Developer's Guide
DataBlade module development tools	*IBM Informix DataBlade Developer's 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 Programmer's Manual
	IBM Informix GLS User's Guide
	IBM Informix CLI Programmer's Manual
	IBM Informix ESQL/C Programmer's Manual
IBM Informix Dynamic Server	*IBM Informix Dynamic Server Getting Started Guide*
	IBM Informix Dynamic Server Administrator's Guide
	IBM Informix Dynamic Server 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 manuals mentioned in the *DataBlade Module Development Overview* are listed alphabetically by title, with a brief description of each.

Manual Title	Description
J/Foundation Developer's Guide	Describes how Java is implemented in the 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 Informix DataBlade Developer's Kit User's Guide	Describes how to develop and package DataBlade modules 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.

(1 of 3)

Manual Title	Description
IBM Informix R-Tree Index User's Guide	Describes the IBM 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 a new operator class that uses the R-tree access method to index a user-defined data type.
IBM Informix Dynamic Server Getting Started Guide	Provides an overview of the architecture of your Informix database server, introduces the major features of the server, introduces the server kits, and provides information to help you use the documentation that is included with each kit.
IBM Informix GLS Programmer's Manual	Describes IBM Informix GLS, an application programming interface available in IBM Informix products. IBM Informix 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 GLS User's Guide	Describes the Global Language Support (GLS) feature available in IBM Informix products. The GLS feature allows IBM Informix APIs and Informix database servers to handle different languages, cultural conventions, and code sets. This manual describes only the language-related topics that are unique to GLS.
IBM Informix CLI Programmer's Manual	Explains how to use the IBM Informix CLI application programming interface to gain access to Informix databases, manipulate the data in your program, interact with the database server, and check for errors.

(2 of 3)

Manual Title	Description
IBM Informix ESQL/C Programmer's Manual	Explains how to use IBM Informix ESQL/C to create client applications with database management capabilities. This manual 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 manual 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 Dynamic Server Administrator's Guide	Describes how to install, configure, and use the features of your Informix database server.
IBM Informix Dynamic Server Performance Guide	Explains how to configure and operate your database server to improve overall system performance as well as the performance of SQL queries.
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.

(3 of 3)

Topic-to-Title Reference

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

Topic	Detail/Manual Title
ActiveX value objects	*IBM Informix DataBlade Developer's Kit User's Guide*
Aggregates	*IBM Informix DataBlade API Programmer's Guide* *IBM Informix DataBlade Developer's Kit User's Guide*
APIs	DataBlade API: *IBM Informix DataBlade API Programmer's Guide* IBM Informix CLI: *IBM Informix CLI Programmer's Manual* IBM Informix ESQL/C: *IBM Informix ESQL/C Programmer's Manual* IBM Informix GLS: *IBM Informix GLS Programmer's Manual* IBM Informix JDBC Driver: *IBM Informix JDBC Driver Programmer's Guide*
BladeManager	*IBM Informix DataBlade Module Installation and Registration Guide* *IBM Informix DataBlade Developer's Kit User's Guide*
BladePack	*IBM Informix DataBlade Developer's Kit User's Guide*
BladeSmith	*IBM Informix DataBlade Developer's 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 Informix DataBlade Developer's Kit User's Guide*
Compiling source code	*IBM Informix DataBlade Developer's Kit User's Guide*

(1 of 4)

Topic	**Detail/Manual Title**
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 Informix DataBlade Developer's Kit User's Guide*
Dependencies between DataBlade modules	*IBM Informix DataBlade Module Installation and Registration Guide*
Error messages	*IBM Informix DataBlade Developer's Kit User's Guide* *IBM Informix DataBlade API Programmer's Guide*
Example DataBlade modules	DBDK InfoShelf
Java value objects	*IBM Informix DataBlade Developer's Kit User's Guide*
Importing files	*IBM Informix DataBlade Developer's Kit User's Guide*
Inheritance	*IBM Informix Guide to SQL: Tutorial*
Installing DataBlade modules	*IBM Informix DataBlade Module Installation and Registration Guide* *IBM Informix DataBlade Developer's Kit User's Guide*
Interfaces to DataBlade modules	*IBM Informix DataBlade Developer's Kit User's Guide* DBDK Tutorial
Internationalization	*IBM Informix GLS Programmer's Manual*, *IBM Informix GLS User's Guide*
Memory management	*IBM Informix DataBlade API Programmer's Guide*

(2 of 4)

Topic	Detail/Manual Title
Operator class support functions	*IBM Informix User-Defined Routines and Data Types Developer's Guide* *IBM Informix R-Tree Index User's Guide* IBM Informix Developer Zone at www.ibm.com/software/data/developer/informix
Packaging a DataBlade module	*IBM Informix DataBlade Developer's Kit User's Guide*
Performance issues	*IBM Informix Dynamic Server 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 Informix DataBlade Developer's Kit User's Guide*
Secondary access methods	*IBM Informix Dynamic Server Performance Guide* *IBM Informix R-Tree Index User's Guide* IBM Informix Developer Zone at www.ibm.com/software/data/developer/informix
Server architecture	*IBM Informix Dynamic Server Administrator's Guide* *IBM Informix Dynamic Server Getting Started Guide*
Shared memory	*IBM Informix DataBlade Developer's Kit User's Guide* *IBM Informix Dynamic Server Administrator's Guide*

(3 of 4)

Topic	Detail/Manual Title
Smart large objects	*IBM Informix DataBlade API Programmer's Guide* *IBM Informix Large Object Locator DataBlade Module User's Guide* *IBM Informix Guide to SQL: Tutorial* IBM Informix Developer Zone at www.ibm.com/software/data/developer/informix
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 Dynamic Server Administrator's Guide*
Stored Procedure Language	*IBM Informix User-Defined Routines and Data Types Developer's Guide* *IBM Informix Guide to SQL: Syntax*
Testing and debugging DataBlade modules	*IBM Informix DataBlade Developer's Kit User's Guide*
Unit tests	*IBM Informix DataBlade Developer's Kit User's Guide*
Virtual processors	*IBM Informix DataBlade Developer's Kit User's Guide* *IBM Informix Dynamic Server Administrator's Guide*

(4 of 4)

IBM Informix DataBlade Modules

This appendix introduces some of the IBM Informix DataBlade modules to serve as examples of the type of functionality DataBlade modules can provide. For more information on available DataBlade modules, see the IBM Informix Developer Zone at www.ibm.com/software/data/developer/informix.

This appendix includes descriptions of the following DataBlade modules and the extensions to IBM Informix Dynamic Server that each one provides:

- IBM Informix Geodetic DataBlade Module
- IBM Informix Large Object Locator DataBlade Module
- Excalibur Text Search DataBlade Module
- IBM Informix TimeSeries DataBlade Module
- IBM Informix Video Foundation DataBlade Module
- IBM Informix Web DataBlade Module

IBM Informix Geodetic DataBlade Module

The IBM Informix Geodetic DataBlade module manages objects defined on the Earth's surface with a high degree of precision. It is designed to manage spatio-temporal data with global content, such as metadata associated with satellite images. To do this, the IBM Informix Geodetic DataBlade module module uses a latitude and longitude coordinate system on an ellipsoidal Earth model, or geodetic datum, rather than a planar, *x*- and *y*-coordinate system.

With the IBM Informix Geodetic DataBlade module, you can extend your database server to store and manipulate objects in space, referenced by latitude and longitude. You can also provide additional attributes representing an altitude range and a time range.

You can create the following spatio-temporal objects with this module:

- Points
- Line segments
- Strings
- Rings
- Polygons
- Boxes
- Circles
- Ellipses

Extensions to Dynamic Server

The IBM Informix Geodetic DataBlade module provides three kinds of data types:

- **Spatio-temporal.** To represent objects.
- **Distinct and range.** Building blocks that simplify the construction and manipulation of spatio-temporal objects.

Spatio-temporal data types enable you to store data that represents spatial objects with a time component in an Informix database. The time component associates the object with a time period or moment in time. Spatio-temporal types are defined in a type hierarchy, with a supertype—GeoObject—that has the time and altitude attributes common to all spatio-temporal types.

Distinct types provide angular components such as latitude and longitude and linear components such as distance.

Range data types specify the altitude and time dimensions of spatio-temporal data.

The IBM Informix Geodetic DataBlade module provides more than 60 functions to enable you to manipulate your spatio-temporal data. These functions can be grouped into the following categories:

- **Accessor functions.** These functions allow you to access information about objects stored in a database. For example, the **Latitude** function returns—or accesses—the latitude value of an object.

  Type verification functions are included in this group.
- **Computational functions.** These functions perform standard computations.
- **Constructor and conversion functions.** These functions are the basis for the data types. Many of these functions are overloaded, with a conversion syntax in addition to a constructor. The conversion syntax converts an object from the GeoObject supertype back to its original representation.
- **Spatial operators.** The functions take spatio-temporal objects and test them for proximity or intersection. Spatial operators return Boolean `TRUE` or `FALSE`.
- **Data validation functions.** These functions verify GeoRing and GeoPolygon objects.
- **System functions.** These functions control session parameters and provide information about the version of the IBM Informix Geodetic DataBlade module.
- **Tracing functions.** These functions control the trace files and tracing output.
- **Warning functions.** These functions control warning messages and output.
- **Z Value functions.** These functions manipulate Z values.

The IBM Informix Geodetic DataBlade module also provides:

- a client utility called **geovalidate** that you can use to verify the correctness of your input data.
- support for the R-tree spatial access method. The R-tree access method allows you to create an index on columns containing spatio-temporal data.

Documentation

For more information, see the *IBM Informix Geodetic DataBlade Module User's Guide*.

IBM Informix Large Object Locator DataBlade Module

The IBM Informix Large Object Locator DataBlade module enables you to create a single consistent interface to smart large objects. It expands the concept of smart large objects to include data stored outside the database. Smart large object data exceeds a length of 255 bytes or contains non-ASCII characters.

With the IBM Informix Large Object Locator DataBlade module, you create a reference to a smart large object and store the reference as a row in the database. The object itself can reside outside the database: for example, on a file system (or it could be a BLOB or CLOB column in the database). The reference identifies the type, or access protocol, of the object and points to its storage location.

The IBM Informix Large Object Locator DataBlade module is a foundation DataBlade module for other DataBlade modules that handle smart large objects.

Extensions to Dynamic Server

The IBM Informix Large Object Locator DataBlade module provides new data types and functions to extend your database server.

The **lld_locator** data type is a row data type that contains the access protocol for a smart large object and a pointer to its location. The **lld_lob** data type is an opaque data type that is identical to the BLOB and CLOB data types, except that in addition to pointing to the data, it tracks whether the underlying smart large object contains binary or character data. It is contained in the **lld_locator** data type as a field in the row.

The IBM Informix Large Object Locator DataBlade module provides basic functions for creating, opening, closing, deleting, reading from, and writing to smart large objects; client functions to process client files; utility functions for raising errors; and functions for copying smart large objects.

Most of the IBM Informix Large Object Locator DataBlade module functions are implemented in an API library, an ESQL/C library, and an SQL interface for maximum programming flexibility.

Documentation

For more information, see the *IBM Informix Large Object Locator DataBlade Module User's Guide*.

Excalibur Text Search DataBlade Module

The Excalibur Text Search DataBlade module 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 Excalibur Text Search DataBlade module can search any type of text.

The 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 specifically designed to perform sophisticated and fast text searches. It runs in one of the database server-controlled virtual processes.

Extensions to Dynamic Server

The Excalibur Text Search DataBlade module 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 manually convert them from their proprietary format into ASCII when creating an **etx** index; the Excalibur Text Search DataBlade module does this for you. One of the components of the 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 Excalibur Text Search DataBlade module supplies several routines that you can use to perform tasks such as creating and dropping synonym and stopword lists.

Documentation

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

IBM Informix TimeSeries DataBlade Module

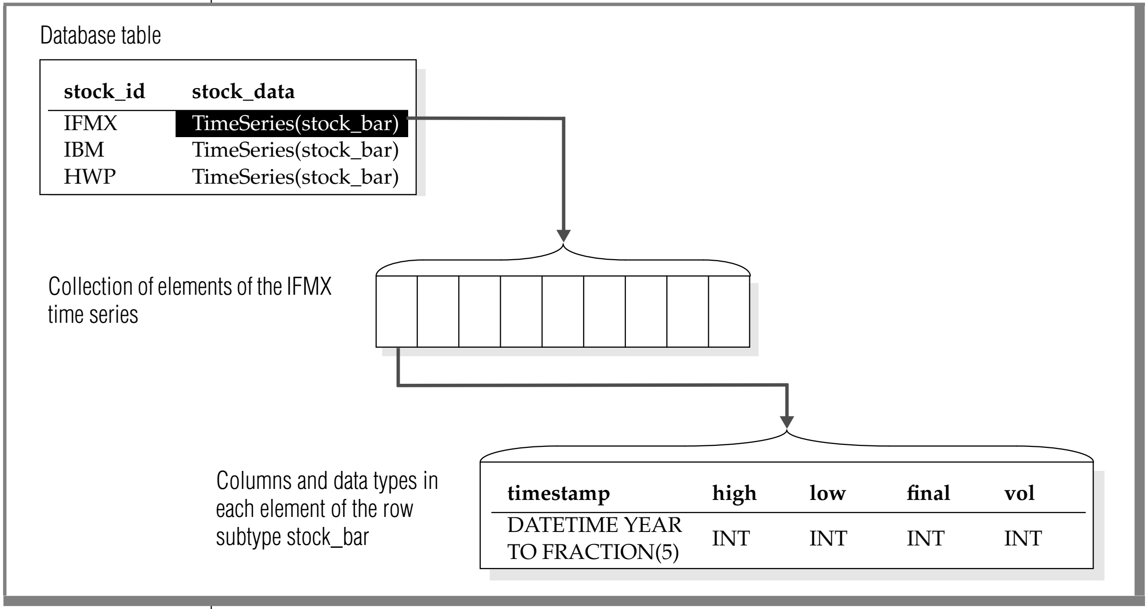
The IBM Informix TimeSeries DataBlade module enables you to store and manipulate a series of data entries associated with a date and time. This timestamped data is stored in a row type, which you define to include whatever data you want, in addition to the timestamp. You also control the granularity of time recording. The IBM Informix TimeSeries DataBlade module supports regularly or irregularly repeating timestamped series.

Time series data is stored and analyzed by applications in many different industries, including manufacturing, journalism, science, and engineering. Time series data is also used in the financial world for corporate financial reporting, stock prices, bond yields, and derivative securities.

Extensions to Dynamic Server

The IBM Informix TimeSeries DataBlade module allows you to create an intuitively organized data model by grouping all the timestamped data for an entity into a single row in a database table, using the **TimeSeries** data type. The **TimeSeries** data type is a type constructor that creates a collection of elements that are row types, as illustrated in Figure B-1.

Figure B-1
Time Series Architecture


Database table
stock_id
stock_data
IFMX
TimeSeries(stock_bar)
IBM
TimeSeries(stock_bar)
HWP
TimeSeries(stock_bar)
Collection of elements of the IFMX time series
Columns and data types in each element of the row subtype stock_bar
timestamp
high
low
final
vol
DATETIME YEAR TO FRACTION(5)
INT
INT
INT
INT

You create the row type to fit your data; the first field, or column, in the row must be a timestamp of type DATETIME YEAR TO FRACTION 5, but the rest of the columns can be any data type supported in row types. Since the time series data is in a row type instead of an opaque type, you can retrieve individual columns within an element.

The collection elements are indexed according to their timestamp, making retrieval of chronologically contiguous elements very fast. Once the number of elements exceeds the user-supplied threshold, the IBM Informix TimeSeries DataBlade module moves all the elements to a container, which exists in a user-defined dbspace. Containers are necessary because time series data typically becomes too large to fit in a database table. Containers also allow you to retrieve only the information you need, instead of the whole time series, as would happen if you used a smart large object to store your data.

For both regular and irregular time series, the **Calendar** and **CalendarPattern** data types allow you to specify an arbitrarily complex pattern of when entries are accepted. For regular time series, the calendar additionally creates a vector from which to calculate an element's position, so that an element's timestamp does not have to be stored, or even specified.

The IBM Informix TimeSeries DataBlade module provides a wide variety of routines to manage and manipulate time series data. The routines allow you to manipulate columns in an element, an element itself, a portion of a time series, a whole time series, or many time series. In addition, there are routines to manage calendars, containers, and metadata. Most routines are implemented in both an SQL interface and an API library.

The IBM Informix TimeSeries DataBlade module also includes system tables to record information about calendars, time series, and containers.

Documentation

For more information, see the *IBM Informix TimeSeries DataBlade Module User's Guide*.

IBM Informix Video Foundation DataBlade Module

The IBM Informix Video Foundation DataBlade module enables you to store, manage, and manipulate *video data* and *metadata* in the same system in which you store more traditional data. Such a system is called a *media management system*.

Video data represents an image that changes continuously in time. Unlike traditional data that is retrieved and displayed as a discrete value, video data is incrementally accessed and is displayed as a real-time stream of frames. Video data can be located on one or more storage servers, such as Web video streamers, VTRs (video tape recorders), or video servers. If an IBM Informix Video Foundation interface has been created for the storage server, that storage server can come under database control.

Video metadata is data about the video data. This is the information that is actually stored in database tables, along with information on how to retrieve requested video segments from the various storage servers. The Video Foundation DataBlade module defines two categories of metadata:

- **Physical attributes.** Descriptions of the media on which the video data is stored, such as length, format, and capture rate.
- **Abstract descriptions.** Multimedia descriptions of the video data content, such as text annotations, or audio or video clips that represent a larger video program.

The Video Foundation DataBlade module is referred to as a *foundation* DataBlade module because it provides a base on which new or specialized video technologies can be quickly added or changed. Because the foundation is open, secure, and scalable, it provides a clear path toward reusing and repurposing valuable video assets among video formats and distribution channels.

A *media management system* is a collection of processes, computers, and other machines that enable you to store and manipulate video (and, often, other media) data along with more traditional data, such as text. The IBM Informix Video Foundation DataBlade module provides interfaces among the following basic elements of such a media management system:

- A video client application, such as an editing or logging application, running on a stand-alone computer.
- Your Informix database server and the Video Foundation DataBlade module, running on a second computer; the database contains the video metadata. (Other IBM Informix and third-party DataBlade modules—such as IBM Informix Image, Text, and Web DataBlade modules, and third-party storage server and video scene-change detection DataBlade modules—can also be running on this computer.)
- Video data located on one or more storage servers, such as Web video streamers, VTRs, or video servers. Generally, each storage server is hosted on a separate computer.

Extensions to Dynamic Server

The Video Foundation DataBlade module consists of the following components:

- **Temporal component.** This component consists of the data types and functions that provide a format-independent interface between your Informix database server and video data. This interface enables the server to reference specific points in time (*media points* and *timecodes*) and intervals (*media chunks*) within video data, regardless of the data format (AVI, MPEG, and so on).
- **Storage component.** This component consists of the data types and functions that provide a device- and format-independent interface between your Informix database server, video data, and the storage servers on which that data is located.
- **Video database schema.** This schema is a set of database tables that connect the data of the temporal and storage components; it is a repository for video metadata.

- **Client API.** This API provides the opaque types of the Video Foundation DataBlade module as value objects for application developers using the IBM Informix C++ client interface. It also provides a subset of the DataBlade module server functions as client-side functions for application developers using any of the following client APIs:
  - IBM Informix ESQL/C API
  - IBM Informix DataBlade API
  - IBM Informix Object Interface for C++

Documentation

For more information, see the *IBM Informix Video Foundation 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 Informix database.

Using the Web DataBlade module, you need not develop a Common Gateway Interface (CGI) application to dynamically access database data. Instead, you create HTML pages that include 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, as well as HTML, image, audio, and video data.

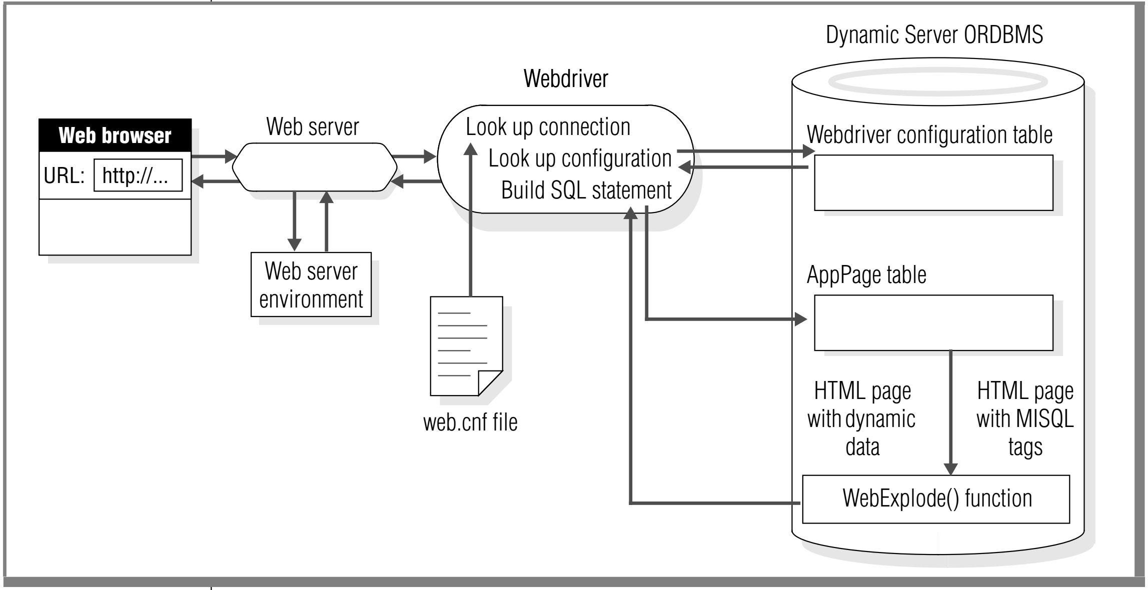
Extensions to Dynamic Server

The Web DataBlade module consists of three main components:

- **Webdriver.** As a client application to your 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 Web DataBlade module tags within HTML and dynamically builds and executes the SQL statements and processing instructions embedded in the 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 using SGML-compliant processing tags.
- **Web DataBlade module tags and attributes.** The 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 Web DataBlade module.

Figure B-2
Web DataBlade Module Architecture



When a URL contains a Webdriver request, the Web browser makes a request to the Web server to invoke 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 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.

Documentation

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

Notices

Trademarks

Glossary

access method	A set of server routines that the database server uses to access and manipulate an index or a table. B-tree is the default secondary access method used by DataBlade modules. Some DataBlade modules have their own access methods, with routines defined by the module. See also *primary access method*, *secondary access method*.
access privilege	Permission for a user to perform an operation in a specific database, table, table fragment, or column.
ActiveX value object	A Microsoft Common Object Model (COM)-compliant object that contains a client-side copy of an opaque type and its support routines. See also *value object*.
aggregate function	A function that performs a mathematical operation on a set of rows selected by a query and returns a single value that contains information about those rows. Aggregates use one or more support functions to perform the aggregate operations. Examples of built-in aggregates include SUM, AVG, and COUNT.
arithmetic function	A function that returns a value by performing a mathematical operation on one or more arguments.
B-tree index	A type of index that uses a balanced tree structure for efficient record retrieval. B-tree indexes store key data in ascending or descending order.
BLOB	A smart large object data type that stores any kind of binary data, including images. The database server performs no interpretation on the contents of a BLOB column.

	See also *smart large object*.
built-in cast	A cast that is built into the database server. A built-in cast performs automatic conversions between different built-in data types.
built-in data type	A fundamental data type defined by the database server: for example, INTEGER, CHAR, or SERIAL8.
built-in function	A predefined, SQL-invoked function that provides some basic arithmetic and other operations, such as **cos()**, **log()**, or **today()**.
cast	A mechanism that the database server uses to convert data from one data type to another. The server provides built-in casts that it performs automatically. Users can create both implicit and explicit casts. See also *cast support function*, *explicit cast*, *implicit cast*, *built-in cast*.
cast support function	A function that is used to implement an implicit or explicit cast by performing the necessary operations for conversion between two data types. A cast support function is optional unless the internal storage representations of the two data types are not equivalent.
class	A category of objects that have common properties and are managed through a specific system table. Informix database classes include access methods, aggregates, casts, routines, operators, tables, and types.
client files	The files that reside on a client workstation that accesses a DataBlade module's objects. Not all DataBlade modules have client files. Examples include client applications or client libraries that are specific to the DataBlade module.
CLOB	A smart large object data type that stores blocks of text items, such as ASCII or PostScript files. See also *smart large object*.
collection	An instance of a collection data type; a group of elements of the same data type stored in a SET, MULTISET, or LIST type constructor. See also *collection data type*.
collection data type	A complex data type that groups values, called elements, of a single data type in a column. Collection data types consist of the SET, MULTISET, or LIST type constructor and an element type, which can be any data type, including a complex data type.

commutator function	A Boolean function that accepts the same two arguments, in reverse order, as another Boolean function, and returns the same result. The query optimizer might choose the commutator function if it executes more quickly in a given query than the specified function. See also *negator function*.
complex data type	A data type that is built from a combination of other data types using an SQL type constructor or the CREATE ROW TYPE statement, and whose components can be accessed through SQL statements. Complex data types include collection data types and row data types.
concurrency	The ability of two or more processes to access the same database simultaneously.
constructed data type	A complex data type created with a type constructor: for example, a collection data type or an unnamed row data type.
constructor	See *type constructor*.
data file	A flat file containing data to be loaded into the database.
data type	See *built-in data type*, *extended data type*.
database object	A discrete entity within a database, such as a data type, a routine, a table, an index, or a view. Users can define database objects with the CREATE statement.
DataBlade API	The C application programming interface (API) for your Informix database server. The DataBlade API is used for the development of DataBlade modules. The DataBlade API contains routines to process data in the database server and return the results to the calling application.
DataBlade module	A group of database objects and supporting code that extends an object-relational database to manage new kinds of data or add new features. A DataBlade module can include new data types, routines, casts, access methods, SQL code, client code, and installation programs.
distinct data type	A data type based on an existing opaque, built-in, distinct, or named row data type, which is known as its source type. The distinct data type has the same internal storage representation as its source type, but it has a different name. To compare a distinct data type with its source type requires an explicit cast. A distinct data type inherits all routines that are defined on its source type.

element	A member of a collection. See also *collection data type*.
element data type	The data type of the elements in a collection.
explicit cast	A cast that requires a user to specify the CAST AS keyword or cast operator (::) to convert data from one data type to another. See also *cast*, *cast support function*.
extended data type	A data type that is not built-in: namely, a collection data type, row data type, opaque data type, or distinct data type.
external function	An external routine that can accept one or more arguments and returns a single value.
external procedure	An external routine that can accept one or more arguments, but does not return a value.
external routine	A routine written in a language external to the database (for example, C), whose body is stored outside the database but whose name and parameters are registered in the system catalog tables.
field	A component of a named row data type. A field has a name and a data type and is accessed in an SQL statement by using dot notation: for example, *row_type_name.field_name*.
function	A routine that can accept arguments and returns one or more values. See also *built-in function*, *routine*, *user-defined function*.
functional index	An index that stores the result of executing a specified function on a table column.
function overloading	See *routine overloading*.
Global Language Support (GLS)	An application environment that allows IBM Informix application-programming interfaces (APIs) and database servers to handle different languages, cultural conventions, and code sets. Developers use the GLS libraries to manage all string, currency, date, and time data types in their code. Using GLS, you can add support for a new language, character set, and encoding by editing resource files, without access to the original source code and without rebuilding the DataBlade module or client software.

grant privileges	Privileges granted to one or more users. The users then have the authority to grant these same privileges to other users. A privilege list identifies the exact privileges to be granted.
hash rule	A user-defined algorithm that maps each row in a table to a set of hash values used to determine the fragment in which a row is stored.
implicit cast	A cast that the database server automatically performs to convert data from one data type to another. See also *cast*, *cast support function*.
index	A structure of pointers to rows of data in a table. An index optimizes the performance of database queries by ordering rows to make access faster.
Informix user ID	A login user ID (login user name) that must be valid on all computer systems (operating systems) involved in the client's database access. Often referred to as the client's "user ID" or "user name."
Informix user password	A user ID password that must be valid on all computer systems (operating systems) involved in the client's database access. When the client specifies an explicit user ID, most computer systems require the Informix user password to validate the user ID.
INFORMIXDIR	The Informix environment variable that specifies the directory in which IBM Informix products are installed.
inheritance	The property that allows a named row data type or a typed table to inherit representation (data fields and columns) and behavior (routines, operators, and rules) from a named row data type or typed table superior to it in a defined hierarchy. Inheritance allows for incremental modification, so that an object can inherit a general set of properties and then add properties that are specific to itself. Under certain circumstances, distinct data types can also have inheritance.
input parameter	A placeholder within a prepared SQL statement that represents a value to be provided at the time the statement is executed.
interface	In the DataBlade Developers Kit, a way to refer to a DataBlade module within another DataBlade module. Because an interface creates a dependency on another module, BladeManager ensures that the originating module is registered before the module that contains the interface.

iterator function A function that returns a set of results one row at a time. The database server calls iterator functions repeatedly to process all the return values.

keyword A word that has meaning to a programming language. In Informix SQL, keywords are shown in syntax diagrams in all uppercase letters. They must be used in SQL statements exactly as shown in the syntax, but they can be entered as either uppercase or lowercase letters.

large object A data object that exceeds 255 bytes in length. A large object is logically stored in a table column but physically stored independently of the column, because of its size. Large objects can contain non-ASCII data. Your Informix database server recognizes two kinds of large objects: simple large objects (TEXT, BYTE) and smart large objects (CLOB, BLOB).

See also *SLV*, *smart large object*.

LIST constructor A type constructor used to create a LIST data type.

LIST data type A collection data type in which elements are ordered and duplicates are allowed.

See also *collection data type*.

locale A set of files that define the native-language behavior of the program at runtime. The rules are usually based on the linguistic customs of the region or the territory. The locale can be set through an environment variable that dictates output formats for numbers, currency symbols, dates, and time, as well as collation order for character strings and regular expressions.

See also *Global Language Support (GLS)*.

LVARCHAR A built-in data type that stores varying-length character data greater than 256 bytes. It is used for input and output casts for opaque data types. LVARCHAR supports code-set order for comparisons of character data.

math function See *built-in function*, *operator function*.

member A component of an opaque data type. A member has a name and a data type and can be accessed in an SQL statement by user-defined accessor functions.

multirepresentational data type A data type whose storage location can switch between a row and a smart large object.

MULTISET constructor A type constructor used to create a MULTISET data type.

MULTISET data type	A collection data type in which elements are not ordered and duplicates are allowed. See also *collection data type*.
named row data type	A row data type that is created with the CREATE ROW TYPE statement and has a name. A named row data type can be used to construct a typed table and can be part of a type or table hierarchy. See also *row data type*, *unnamed row data type*.
negator function	A Boolean function that accepts the same arguments in the same order as another Boolean function, but returns the Boolean complement. The query optimizer might choose the negator function if it executes more quickly in a given query than the specified function. See also *commutator function*.
nonvariant function	A function that always returns the same value when passed the same arguments.
not null constraint	A constraint on a column that specifies that the column cannot contain null values.
object	See *database object*.
objects script	A file containing SQL statements that describe the objects in a DataBlade module.
opaque data type	An extended data type that contains one or more members but whose internal structure is interpreted by the database server using user-defined support routines.
operator	A symbol, such as =, >, +, or -, that invokes an operator function.
operator binding	The association between an operator and an operator function. Operator binding occurs when an SQL statement contains an operator and the database server automatically invokes the associated operator function.
operator class	The set of operators that the database server associates with a secondary access method. When an index is created, it is associated with a particular operator class.

operator function	An arithmetic function that has a corresponding operator symbol. An operator function processes one to three arguments and returns a value. For example, the **plus()** function corresponds to the "+" operator symbol.
overloading	See *routine overloading*.
parallel database query (PDQ)	A query that allows the database server to distribute the execution of the query among several virtual processors by dividing it into subqueries.
parallelizable routine	A routine that can be executed within a parallel database query statement. See also *parallel database query (PDQ)*.
parameter	A variable to which a value can be assigned in a specific application. In a routine, a parameter is the placeholder for the argument values passed to the subroutine at runtime.
polymorphism	See *routine overloading*.
prepare script	A file containing SQL statements that describe the DataBlade module. There are two types of prepare scripts: ■ The script called **prepare.sql** contains information about the module that is not language-specific. ■ Scripts with names in the format **prepare.*locale*.sql** contain language-specific information such as the module and vendor descriptions.
primary access method	A set of routines that perform table operations such as inserting, deleting, updating, and scanning data. The database server provides a virtual table interface (VTI), with which advanced users can create primary access methods for virtual tables.
privilege	Rights granted to specific users on specific objects within the database. A privilege list identifies the exact privileges that are applicable for a particular object and that are held by the user invoking the grant. Privileges are granted or revoked on a database object using the GRANT and REVOKE statements.
procedure	A routine that can accept arguments but does not return a value. See also *external procedure*, *stored procedure*.
procedure overloading	See *routine overloading*.

query optimizer	A server facility that estimates the most efficient plan for executing a query in the DBMS. The optimizer considers the CPU cost and the I/O cost of executing a plan.
R-tree index	A type of index that uses a tree structure based on overlapping bounding rectangles to speed access to spatial and multidimensional data types.
registration	The process of executing SQL statements to create DataBlade module objects or individual user-defined routines in a database and giving the database server the location of the associated shared object file. Registration makes a DataBlade module available for use by client applications that open that database.
routine	A named collection of program statements that perform a particular task and can accept arguments. Routines include functions, which return one or more values, and procedures, which do not return values. See also *function*, *procedure*, *user-defined routine*.
routine resolution	The process that the database server uses to determine which routine to execute, given the routine signature. See also *routine signature*.
routine signature	The information that the database server uses to identify a routine. The signature of a routine includes the type of the routine (function or procedure), the routine name, the number of parameters, the data types of the parameters, and the order of the parameters. In an ANSI-compliant database, the name of the routine is specified as *owner.name*.
routine overloading	Defining more than one routine with the same name but different parameter lists.
ROW constructor	The type constructor used to construct unnamed row data types.
row data type	A complex data type consisting of a group of ordered data elements (fields) of the same or different data types. The fields of a row type can be of any supported built-in or extended data type, including complex data types, except SERIAL and SERIAL8 and, in certain situations, TEXT and BYTE. There are two kinds of row data types: ■ Named row types, created using the CREATE ROW TYPE statement ■ Unnamed row types, created using the ROW constructor

	See also *named row data type*, *unnamed row data type*.
sbspace	A logical storage area that contains one or more chunks that store only smart large object data.
secondary access method	A set of database server functions that build, access, and manipulate an index structure: for example, a B-tree, an R-tree, or an index structure provided by a DataBlade module. Typically, a secondary access method speeds up the retrieval of data.
	When an SQL query uses an index created on a secondary access method, it accesses the index using the functions defined in the operator class associated with that access method.
	See also *operator class*.
selectivity function	A function that calculates the percentage of rows that will be returned by a filter function in the WHERE clause of a query. The optimizer uses selectivity information to determine the fastest way to execute an SQL query.
SET constructor	A type constructor used to create a SET data type.
SET data type	A collection data type in which elements are not ordered and duplicates are not allowed.
	See also *collection data type*.
shared memory	A portion of main memory that processes can use to communicate and share common data, thus reducing disk I/O and improving performance.
signature	See *routine signature*.
SLV	Abbreviation for *statement local variable*.
smart large object	A large object that:

- is stored in an sbspace, a logical storage area that contains one or more chunks.
- has read, write, and seek properties similar to a UNIX file.
- is recoverable.
- obeys transaction isolation modes.
- can be retrieved in segments by an application.

Smart large objects include CLOB and BLOB data types.

SPL	Abbreviation for *Stored Procedure Language*.
statement local variable (SLV)	A variable for storing a value that a function returns indirectly, through a pointer, in addition to the value that the function returns directly. An SLV's scope is limited to the statement in which it is used. The *RANK* parameter of the **Resembles** operator function is an SLV.
stored procedure	A user-defined routine that is stored in a database in executable format. Stored procedures are used to execute frequently repeated tasks, to improve performance, and to monitor access to data. Stored procedures are written in Stored Procedure Language (SPL).
Stored Procedure Language	An Informix extension to SQL that provides flow-control features such as sequencing, branching, and looping, comparable to those features provided in the SQL/PSM standard. SPL can be used for writing DataBlade module routines.
strategy functions	The functions that the optimizer uses to determine what filters in a query can use a secondary access method (index).
subquery	A SELECT statement within a WHERE clause.
support routines	The internal routines that the database server automatically invokes to process a data type, cast, aggregate, or access method. The database server uses user-defined support routines to perform operations on opaque data types (such as converting to and from the internal, external, and binary representations of the type). An secondary access method uses a support routine in an operator class to perform operations on an index (such as building or searching).
system catalog	A group of database tables that contain information about the database itself, such as the names of tables or columns in the database, the number of rows in a table, the information about indexes and database privileges, and so on.
table	A rectangular array of data in which each row describes a single entity and each column contains the values for each category of description. A table is sometimes referred to as a *base table* to distinguish it from the views, indexes, and other objects defined on the underlying table or associated with it.

type constructor	An SQL keyword that indicates to the database server the type of complex data to create. See also *LIST constructor*, *MULTISET constructor*, *ROW constructor*, *SET constructor*.
type inheritance	The property that allows a named row data type to inherit representation (data fields, columns) and behavior (routines, operators, rules) from a named row type above it in the type hierarchy.
unnamed row data type	A row type created with the ROW constructor that has no defined name and no inheritance properties. Two unnamed row types are equivalent if they have the same number of fields and if corresponding fields have the same data type, even if the fields have different names.
unregistration	The process of executing SQL statements to drop DataBlade module objects or individual user-defined routines in a database and removing the ability to access the associated shared object file from the database server.
user-defined function	A user-defined routine that returns a value.
user-defined routine	A routine, written in one of the languages that your Informix database server supports, that provides added functionality for data types or encapsulates application logic.
user-defined procedure	A user-defined routine that does not return a value.
user-defined statistics	Information about the opaque data type values in your database that is collected by the UPDATE STATISTICS statement, which calls user-defined functions to calculate the statistics. The optimizer uses these statistics to determine the fastest way to execute an SQL query.
user-defined virtual processor	A virtual processor that executes the user-defined routines that are assigned to it. See also *virtual processor*.
value object	A self-contained binary object that provides standard interfaces to its callers. Value objects can be used in client applications.

variant function	A function that, with the same arguments, can either return different values or have varying side effects, such as updating a table or external file.
virtual processor	One of the multithreaded processes that make up the database server and are similar to the hardware processors in the computer.

Index