

About the DataBlade Developers Kit Tutorial

The DataBlade Developers Kit Tutorial is a set of exercises that demonstrate creating extensions to Informix Dynamic Server and packaging them as DataBlade modules using the DataBlade Developer Kit.

Note: You should have a basic understanding of extensibility before you start this tutorial. The focus of this tutorial is on creating extensions, not on defining them. To learn about extensibility, read the "DataBlade Module Development Overview."

The DataBlade Developers Kit consists of three tools:

- **BladeSmith.** For defining the objects that your DataBlade module creates and generating code templates for them
- **BladePack.** For packaging a final installable version of a DataBlade module
- **BladeManager.** For registering DataBlade modules in databases

In the first exercise, AddInts, you learn how to use BladeSmith and BladeManager by creating a simple DataBlade module. All tutorial users should start with Exercise 1.

Exercises 2 through 6 demonstrate how to create DataBlade modules and debug them. Each exercise is more complex than the previous; you can choose to either work through the exercises sequentially or pick just the ones that interest you.

The tutorial contains the following exercises:

1. [Creating a Simple User-Defined Routine](#)
2. [Debugging a DataBlade Module](#)
3. [Creating Distinct Data Types and Casts](#)
4. [Creating Row Data Types](#)
5. [Creating Opaque Data Types](#)
6. [Using Interfaces](#)
7. [Using Smart Large Objects](#)

Tips

At the beginning and end of each exercise you can use a drop-down list of exercises, giving you access to any of the other exercises. You can also click buttons to go to the InfoShelf home page, to this page, or to print this tutorial.

Sample code is shown in blue, like the following fragment:

```
Gen_ReturnVal = Int1 + Int2;
```

When adding code to your source files, you can cut and paste code from this tutorial.

This tutorial has links to C source code (.c) files and SQL (.sql) files. How these files are displayed depends on your setting in Windows Explorer. For example, if you have .c files set to open in Microsoft Developer Studio, then when you click a link to a .c file in the tutorial, the file displays in Microsoft Developer Studio. If you do not have .c and .sql files configured to be opened by another application, the files display in your browser. In that case, you must use your browser's **Back** command to return to the tutorial.

Software Dependencies

To use this tutorial, you need the following Informix software:

- The DataBlade Developers Kit, Version 3.7
- Informix Dynamic Server with Universal Data Option, Version 9.14 or 9.13
- The Informix Client Software Developer's Kit

Exercise 2 requires Microsoft Developer Studio Visual C++, Version 4.2 or later.

See the Read Me First for the DataBlade Developers Kit for system requirements.

Related Reading

For an introduction to the Informix database server and the DataBlade module development process, see the *DataBlade Module Development Overview*.

Tutorial Exercise 1

Creating a Simple User-Defined Routine

This exercise demonstrates how to use the tools of the DataBlade Developers Kit-- [BladeSmith](#) and [BladeManager](#)-- to create a DataBlade module with a user-defined routine.

The AddInts DataBlade module is very simple: it creates a function, **AddIntegers()**, which adds two integers and returns the result.

Overview of Steps

You follow these basic steps to create the AddInts DataBlade module:

1. Use BladeSmith to create the AddInts DataBlade module project.
2. Use BladeSmith to create the **AddIntegers()** function.
3. Use BladeSmith to generate the source code and the SQL registration scripts.
4. Add code for the **AddIntegers()** function to the template generated by BladeSmith.
5. Compile your source code using your C compiler.
6. Install your DataBlade module.
7. Start the database server.
8. Create a test database.
9. Register your DataBlade module.
10. Run your DataBlade module.

These steps are fully explained in this exercise.

Tutorial Steps

This exercise consists of 10 steps. It takes approximately 45 minutes to complete.

1

Use BladeSmith to create the AddInts DataBlade module project.

- ✓ Start BladeSmith from the **Informix** program group.
- ✓ Choose **Project→New** and enter the following information in the first page of the New Project wizard.

Update Project Wizard: page 1

This page lets you name your DataBlade module, and assign a version number. You can update this information by selecting the project name in the tree control and selecting Edit/Properties.

The description locale specifies the default GLS code set for your DataBlade module. If the server compatibility box is set to something besides the newest server version, you will see warning messages if you use any incompatible features.

DataBlade name: Object prefix: Description locale: Server compatibility:

Project version numbers (or letters)

Major: Minor: Revision: Release:

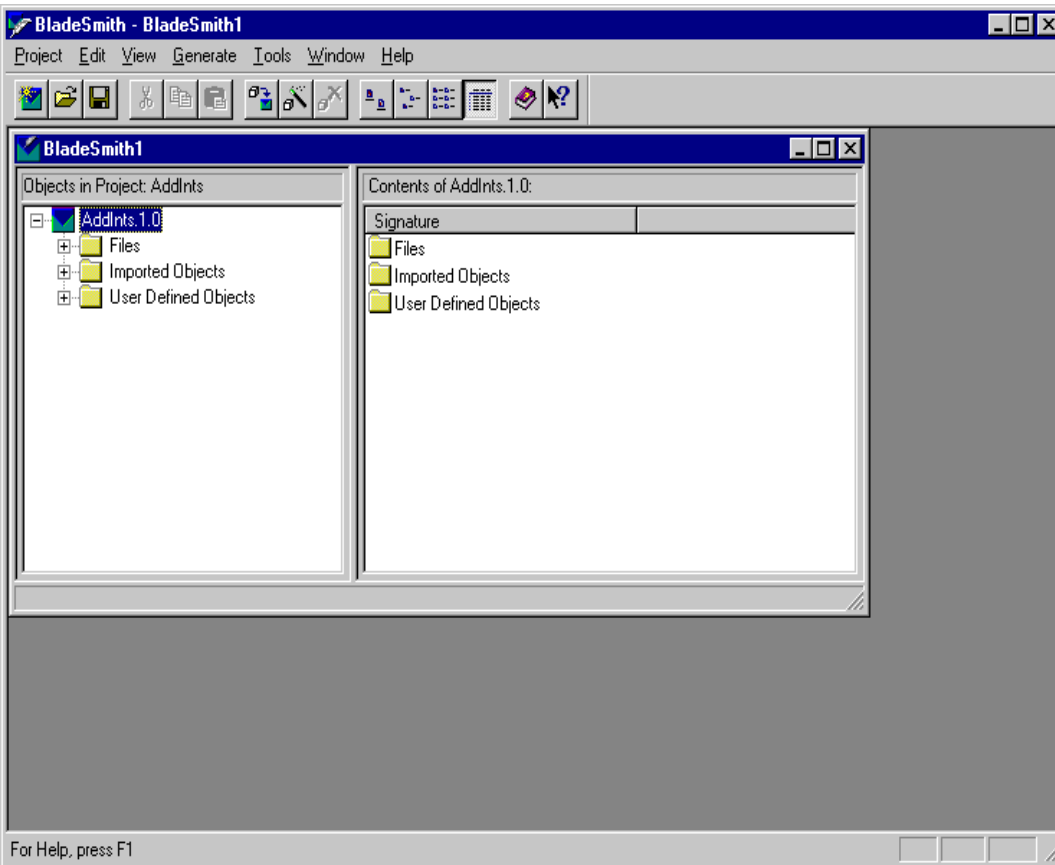
Project description:

The AddInts DataBlade module has one user-defined routine, AddIntegers(), which adds two integers and returns the result.

< Back Next > Finish Cancel Help

The **Object Prefix**, **Description Locale**, and **Server compatibility** fields are mandatory; the rest are optional.

- ✓ Click **Next** and enter information about your company in the second page of the wizard.
- ✓ Click **Next** and the last page of the New Project wizard displays a preview of the SQL generated to define the DataBlade module object to the database server. Most BladeSmith wizards that you use to define a new object display the SQL statements they generate.
- ✓ Click **Finish** to create the project. BladeSmith displays the new project and its folders.



The **Files**, **Imported Objects**, and **User Defined Objects** folders are currently empty, but as you build your DataBlade module, the objects you add appear in this hierarchy.

✓ Choose **Project**→**Save** to save the new DataBlade module. The Save As dialog box opens because this is the first time you are saving this project. Create a directory called **AddInts** in any convenient place on your hard drive and save your project as the file **AddInts.ibs** in this directory.

Exercise 1: Creating a Simple User-Defined Routine, continued

2

Use BladeSmith to create the **AddIntegers()** function.

✓ Choose **Edit→Insert→Routine** and enter the following information in the New Routine wizard.

Routine Property	AddIntegers Value
Routine Name (SQL)	AddIntegers
Return type	integer
Statement Local Variable	No
Arguments	Name: Int1 Type: integer Default: none Name: Int2 Type: integer Default: none
Language	C
Options for how the database server calls this routine (page 5)	Only the Routine does not accept NULL arguments check box should be checked; uncheck the Routine may return inconsistent check box.
Routine Name (C)	AddInts
Shared object path	Leave as default
Behavior of routines	Routine is well-behaved
Special stack requirements	None
Cost	0
Related routines	None
Grant execute privileges	Grant execute privileges

✓ Click **Finish**. The **AddIntegers()** function appears in the **User Defined Objects** folder.

✓ Choose **Project→Save** to save the changes to your DataBlade module.

Exercise 1: Creating a Simple User-Defined Routine, continued

3

Use BladeSmith to generate the source code and the SQL registration scripts.

✓ Choose **Generate→DataBlade**. The Generate DataBlade dialog box appears.

- ✓ Click the **Format** property and choose either DOS or UNIX for the file format, depending on your platform.
- ✓ Select the **DataBlade** node at the top of the file tree and click **Generate DataBlade**.
- ✓ After BladeSmith has finished generating code, click **Close** to close the Generate DataBlade dialog box.

BladeSmith generates the basic C code and [SQL scripts](#) necessary for your DataBlade module:

- The source code files are saved in the **AddInts\src\C** directory.
- The SQL scripts are saved in the **AddInts\scripts** directory.

For more information about the **Generate DataBlade** command, see the *DataBlade Developers Kit User's Guide*.

The BladeSmith portion of this exercise is complete; you can exit BladeSmith.

Exercise 1: Creating a Simple User-Defined Routine, continued

4

Add code for the **AddIntegers()** function to the template generated by BladeSmith.

BladeSmith generates two source code files for the AddInts DataBlade module:

- **udr.c**. For user-defined routines
- **support.c**. For tracing and BladeSmith utility functions (do not edit this file)

The **udr.c** file you just generated contains:

- **#include** statements.
- the **AddIntegers()** function template.

You must add to **udr.c** code that implements the **AddIntegers()** function.

- ✓ To create the final version of **udr.c**, open the file from the **AddInts\src\C** directory and find the following code:

```
/*
** TO DO: Remove this comment and call to
**      mi_db_error_raise after implementing
**      this function.
*/
mi_db_error_raise( Gen_Con, MI_EXCEPTION,
                  Function AddIntegers has not been implemented." );
```

- ✓ Remove the comment and replace the **mi_db_error_raise()** statement with the following code to add the two integers:

```
Gen_RetVal = Int1 + Int2;
```

- ✓ Remove the following comment, the call to the `mi_fp_setreturnisnull()` function, and the statement that sets `GenRetVal` to 0:

```
/*
** TO DO: Compute and store your value into GenRetVal.
** The call to mi_fp_setreturnisnull informs the
** server that the return value is NULL.
*/
mi_fp_setreturnisnull( Gen_fparam, 0, MI_TRUE );
GenRetVal = 0;
```

- ✓ Check your source code against the final version of `udr.c`.
- ✓ Save your changes to `udr.c`.

Exercise 1: Creating a Simple User-Defined Routine, continued

5

Compile your source code using your C compiler.

When you generate source code, BladeSmith creates three [makefiles](#) in the `src\C` directory. The instructions in this exercise use the `AddInts.mak` Microsoft Developer Studio Visual C++ file to compile the source code into the `AddInts.bld` dynamic link library file.

For information about compiling on UNIX, see the *DataBlade Developers Kit User's Guide*.

- ✓ See "[Setting Informix Directories in Microsoft Visual C++](#)" for instructions on configuring Visual C++ with Informix include files and library file directories.
- ✓ Choose **File→Open** and open `AddInts.mak`, your project makefile, from the `AddInts\src\C` directory.
- ✓ Choose **Build→Set Active Configuration**. The Set Active Project Configuration dialog box appears.
- ✓ Select the **Debug** version of the project.
- ✓ Click **OK**.
- ✓ Choose **Build→Build AddInts.bld** to compile.

Microsoft Developer Studio creates a **Debug** subdirectory under the `src\C` directory to hold debugging dynamic link libraries.

6

Install the AddInts DataBlade module.

To package a DataBlade module for installation on other systems, you typically use [BladePack](#). However, because of the simplicity of the AddInts DataBlade module, you do not use BladePack to create an installation package.

- ✓ Create the following new directory on the server machine:

%INFORMIXDIR%\extend\AddInts.1.0

- ✓ Copy the **AddInts.bld** file from the **src\C\Debug** directory and the **.sql** files from the **scripts** directory into the **AddInts.1.0** directory.
- ✓ Ensure that the permissions on the **AddInts.bld** file are read-only. (If your dynamic link library or shared object file is not read-only, a database server error occurs when you try to load it.)

7

Start the database server.

These instructions describe how to start Informix Dynamic Server on Windows NT. For instructions on starting Informix Dynamic Server on UNIX, see the *Administrator's Guide* for your database server.

- ✓ From the Control Panel, open the Services application.
- ✓ Select the INFORMIX-Universal Server service.
- ✓ Click **Start**.

8

Create a test database.

If the database in which you plan to test the AddInts DataBlade module does not yet exist, create it using the following SQL statement:

```
create database addints_test with buffered log;
```

9

Register your DataBlade module in the database.

After you have installed your DataBlade module, you must [register](#) it in the database where you plan to use it.

You should be logged on as the default user for your workstation (so that your NT or UNIX login is the same as your Informix user ID).

- ✓ Start **BladeManager** from the **Informix** program group.
- ✓ In the **Databases** page of the BladeManager window, select the name of the Informix database in which you want to register your DataBlade module.
- ✓ From the **Available** list, select **AddInts.1.0** and click **Add**.
- ✓ Click **Apply** to register the DataBlade module in the database.
- ✓ Click **Exit** to close BladeManager.

See the *DataBlade Module Installation and Registration Guide* for information about using BladeManager from the command line.

10

Run your DataBlade module.

After you install and register the DataBlade module, the Informix database accepts the new **AddIntegers()** function as if it was built-in.

✓ To test **AddIntegers()**, use SQL Editor or DB-Access to connect to the database where you registered your DataBlade module and execute the following SQL statement:

```
execute function AddIntegers(2, 5);
```

The correct result is 7.

✓ Test your DataBlade module with other pairs of numbers to verify that it is functioning correctly.

Tutorial Exercise 2

Debugging a DataBlade Module

This exercise demonstrates debugging a DataBlade module. You will debug the AddInts DataBlade module that you created in Exercise 1. The AddInts DataBlade module contains one user-defined routine, **AddIntegers()**, which adds two integers and returns the result.

This exercise provides instructions for debugging a DataBlade module on Windows NT using Microsoft Visual C++ Developer Studio. You are assumed to be running your Informix server on the Windows NT workstation on which you built the AddInts DataBlade module.

WARNING: Use a test database server for debugging because when you bring down the debugger in Step 5, the database server crashes.

For information about debugging on UNIX, see the *DataBlade Developers Kit User's Guide*. Additional information about debugging is available from the Informix Developers Network.

Tutorial Steps

This exercise consists of seven steps. It takes approximately 30 minutes to complete.

1

Load the DataBlade module by calling one of its routines.

Before you attach the debugger process to the database server, load your DataBlade module dynamic link library file into the database server address space. With the dynamic link library file loaded, you can set breakpoints on module entry points and examine local storage provided by module functions.

✓ To load the DataBlade module into the database server address space, execute a DataBlade module routine using SQL Editor or DB-Access.

Because you have not debugged your routines, you should execute a noncritical routine to load your DataBlade module. There are two methods for loading a DataBlade module dynamic link library without executing an untested critical routine:

- Call the routine with an impossible condition, such as in the following query (you can run this query against any table from which you have permission to select data):

```
select AddIntegers(2,5) from sysbldregistered where 2=0;
```

- Add a simple user-defined routine to your DataBlade module: for example, one that returns a hard-coded version number through SQL. You can then call this simple version number routine instead of a critical routine that you want to test.

2

Start the database server, if necessary.

See "[Starting Informix Dynamic Server on Windows NT](#)" for detailed instructions on starting Informix Dynamic Server on Windows NT.

For instructions on starting Informix Dynamic Server on UNIX, see the *Administrator's Guide* for your database server.

3

Attach the debugger to the server process.

- ✓ Start Microsoft Visual C++ Developer Studio.
- ✓ Choose **Build→Start Debug→Attach to Process**. The Attach to Process dialog box appears.
- ✓ Select the Informix server process **oninit**.
- ✓ Click **OK**.

4

Set breakpoints in your source code.

- ✓ Choose **File→Open** and select **udr.c** from the **AddInts\src\C** directory.
- ✓ To set breakpoints in your code, place your cursor on the line in which you want to set a breakpoint and right-click anywhere on the line. Select **Insert/Remove Breakpoint** from the popup menu.
- ✓ Set a breakpoint at the line in which the **AddInts()** function adds the two integers:

```
Gen_RetVal = Int1 + Int2;
```

- ✓ Save **udr.c**.

5

Issue SQL statements to call your DataBlade module routines.

- ✓ Using SQL Editor or DB-Access, issue the following SQL statements to call the **AddInts()** function:

```
execute function AddIntegers(2,5);
```

The debugger stops the function from executing before it adds the two input values. You can check the values of the variables before and after their values are added. At the breakpoint, the variables have the following values.

Variable Name	Value
Gen_Con	0x0000000
Gen_RetVal	a stack number
Int1	2
Int2	5

✓ Use the **Step Into** or **Step Over** commands in the Debug toolbar to pass the breakpoint. The variables now have the following values.

Variable name	Value
Gen_RetVal	7
Int1	2
Int2	5

6

Stop the debugger and the database server.

- ✓ Run your code until the function is finished executing.
- ✓ In Visual C++, choose **Debug**→**Stop Debugging** and exit Visual C++. The database server crashes.
- ✓ Disconnect SQL Editor or DB-Access from the database. The tool may take a few minutes to exit.

7

If necessary, recompile and repeat the debugging process.

If you have detected problems, you can re-open the Visual C++ workspace for your project (in this example, the file **AddInts.dsw** in the **src\C** directory) and modify the source code to correct the problems. Then save your changes and rebuild your DataBlade module to repeat the debugging process.

Important: You must shut down the database server before you replace the DataBlade module dynamic link library (shared object file). See the "DataBlade Developers Kit User's Guide" for more information.

Tutorial Exercise 3

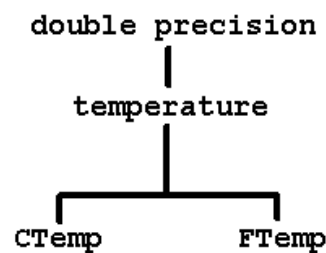
Creating Distinct Data Types and Casts

This exercise demonstrates how to create distinct data types and the casts that convert them.

The Mercury DataBlade module contains [distinct data types](#) to represent Fahrenheit and Celsius temperatures and [casts](#) to convert between the two.

You create the distinct data type hierarchy, beginning with the **temperature** distinct data type, which is based on the built-in double-precision data type.

Then you create the **CTemp** and **FTemp** types that inherit from **temperature**, as shown in the following diagram.



The Mercury DataBlade module ensures that users cannot enter temperatures below absolute zero in the **CTemp** and **FTemp** types. If a user attempts to insert an incorrect value, the INSERT statement fails and returns an error. The DataBlade module uses user-defined support routines and [casts](#) to check the input value.

Using Casts

The Mercury DataBlade module uses casts in two ways:

- To convert temperatures between the Celsius and Fahrenheit.
- To enable users to enter numbers as **integer** and **decimal** data types directly into **CTemp** and **FTemp** columns without having to use an [explicit cast](#) in their SQL statements.

The **temperature** data type is implemented to reduce the number of casts you need to define. If you defined casts from **integer** and **decimal** for both **CTemp** and **FTemp**, you would create a total of six casts. Instead, you add **integer** and **decimal** casts only to the **temperature** type; the **CTemp** and **FTemp** types inherit the casts. This approach reduces the number of casts you have to define from six to four.

DataBlade Module Objects

The Mercury DataBlade module creates the following user-defined objects.

Distinct data types	temperature CTemp FTemp	
Routines	Assign() function for the CTemp data type Assign() function for the FTemp data type	These functions are called whenever the database server assigns a value to a CTemp or FTemp column. They check that the value is greater than absolute zero before assigning it.
Casts (with support routines)	integer to temperature decimal to temperature CTemp to FTemp FTemp to CTemp	These casts convert input data from integer and decimal types to the temperature type. These casts convert values between the CTemp and FTemp types
Error messages	One error message for values less than absolute zero	

Tutorial Steps

This exercise consists of 16 steps. It takes approximately one hour to complete.

1

Create a new project.

- ✓ Start BladeSmith from the **Informix** program group and choose **Project→New**.
- ✓ Enter the following information into the New Project wizard.

Project Property	Mercury Value
DataBlade module name	Mercury
Object prefix	MRC
Description locale	en_us.1252
Server compatibility	9.13 or 9.14
Project version number (<i>Major.Minor</i>)	1.0
Project description	This DataBlade module supports data types and casts for Celsius and Fahrenheit temperatures.
Vendor unique ID (optional)	Your company's name
Company information	Your company's name, copyright, and contact information

- ✓ Click **Finish**.

- ✓ Choose **Project→Save**; create a new directory called **Mercury** in a convenient place on your hard drive and save your project in this directory as a file called **Mercury.ibs**.

Exercise 3: Creating Distinct Data Types and Casts, continued

2

Add the **temperature** distinct data type.

- ✓ Choose **Edit→Insert→Distinct type** and enter the following information in the New Distinct Type wizard.

Distinct Type Property	temperature Value
Distinct type name	temperature
Source type name	double precision
Grant execute privileges	Grant usage privileges

- ✓ Click **Finish**, and then choose **Project→Save** to save the changes to your DataBlade module.

3

Add the **CTemp** and **FTemp** data types.

- ✓ Enter the following information in the New Distinct Type wizard.

Distinct Type Property	CTemp Value	FTemp Value
Distinct type name	CTemp	FTemp
Source type name	temperature	temperature
Grant execute privileges	Grant usage privileges	Grant usage privileges

- ✓ Choose **Project→Save** to save your project.

Exercise 3: Creating Distinct Data Types and Casts, continued

4

Add the user-defined routines.

The Mercury DataBlade module uses two user-defined routines to prevent users from entering values below absolute zero in the data types **CTemp** and **FTemp**. In this DataBlade module, you implement [Assign\(\) functions](#) for the **FTemp** and **CTemp** data types. The C functions that implement the **Assign()** functions are named **FTempAssign()** and **CTempAssign()**. The server calls the appropriate routine before it stores a value for either of the two temperature types.

✓ Choose **Edit→Insert→Routine** and enter the following information, once for each routine, in the New Routine wizard.

Routine Property	FTempAssign Value	CTempAssign Value
Routine Name (SQL)	assign	assign
Return type	FTemp	CTemp
Statement Local Variable	No	No
Arguments	Name: temperature Type: FTemp Default: none	Name: temperature Type: CTemp Default: none
Language	C	C
Options for how the database server calls this routine (page 5)	Only the Routine does not accept NULL arguments check box should be checked; uncheck the Routine may return inconsistent check box.	Only the Routine does not accept NULL arguments check box should be checked; uncheck the Routine may return inconsistent check box.
Routine Name (C)	FTempAssign	CTempAssign
Shared object path	Default	Default
Behavior of routines	Routine is well-behaved	Routine is well-behaved
Special stack requirements	None	None
Cost	0	0
Related routines	None	None
Grant execute privileges	Grant execute privileges	Grant execute privileges

✓ Click **Finish** and then choose **Project→Save** to save your project.

5

Add the **decimal to temperature** cast.

✓ Choose **Edit→Insert→Cast** and enter the following information in the New Cast wizard.

Cast Property	decimal to temperature Value
Cast from type	decimal
Cast to type	temperature
Cast should be called for _implicit type conversion	Yes
Cast requires a user defined cast support function	Yes
Support routine name is:	temperatureCast
Grant execute privileges	Grant execute privileges

✓ Click **Finish** and then choose **Project→Save** to save your project.

Exercise 3: Creating Distinct Data Types and Casts, continued

6

Create the three remaining casts.

✓ Enter the following information in the New Cast wizard.

Cast Property	integer to temperature Value	CTemp to FTemp Value	FTemp to CTemp Value
Cast from type	integer	CTemp	FTemp
Cast to type	temperature	FTemp	CTemp
Cast should be called for_implicit type conversion	Yes	Yes	Yes
Cast requires a user defined cast support function	Yes	Yes	Yes
Support routine name is:	temperatureCast	FTempCast	CTempCast
Grant execute privileges	Grant execute privileges	Grant execute privileges	Grant execute privileges

✓ Choose **Project→Save** to save your project.

Exercise 3: Creating Distinct Data Types and Casts, continued

7

Edit the names of the support routines for the **decimal to temperature** cast and the **integer to temperature** cast.

The support functions for the **decimal to temperature** cast and the **integer to temperature** cast have the same C function name: **temperatureCast()**. SQL allows routines to have the same name as long as they have different parameters, but ANSI C requires unique routine names. Therefore, BladeSmith does not generate source code until you have made these names unique.

✓ To edit the C function name, double-click the support routine that appears under the cast when you expand the hierarchy in the BladeSmith project window. The Properties dialog box appears.

In the Properties dialog box, click the **Language** tab, and then take the following actions:

- For the **decimal to temperature** cast, type **DecToTemp** for the C routine name and click **Apply** and then **OK**.
- For the **integer to temperature** cast, type **IntToTemp** for the C routine name and click **Apply** and then **OK**.

(The default C routine names for the **CTemp to FTemp** cast and the **FTemp to CTemp** cast do not conflict with other routines, so they do not need to be changed.)

- ✓ Choose **Project→Save** to save the changes to your project.

8

Create the error message raised if a user enters a temperature lower than absolute zero.

- ✓ Choose **Edit→Insert→Error** and enter the following information in the New Error wizard.

Error Property	Mercury Error Value
SQL error code	UMRC0
Error locale	en_us.1252
Register message as...	Both
SQL error text	Temperatures cannot go below absolute zero.

- ✓ Click **Finish** and choose **Project→Save** to save your project.

Exercise 3: Creating Distinct Data Types and Casts, continued

9

Generate the source code and the SQL registration scripts.

See "[Generating Code](#)" for detailed instructions.

BladeSmith automatically generates the basic C source code files and the SQL scripts necessary for your DataBlade module to run:

- The source code files are saved in the **Mercury\src\C** directory.
- The SQL scripts are saved in the **Mercury\scripts** directory.

The BladeSmith portion of this exercise is complete; you can exit BladeSmith.

10

Add your source code to the templates generated by BladeSmith.

BladeSmith generates two source code files for the Mercury DataBlade module:

- **udr.c**. For user-defined and cast support functions
- **support.c**. For tracing and BladeSmith utility functions (do not edit this file)

- ✓ Make the following modifications to the source code in **udr.c**:

- Remove the following code from each function:

```

/*
** TO DO: Remove this comment and call to
**      mi_db_error_raise after implementing
**      this function.
*/
mi_db_error_raise( Gen_Con, MI_EXCEPTION,
                  "Function Distance has not been implemented." )

```

- Add code in the section marked `TO DO: Compute and store your value into GenRetVal.`

To find the code you need to add for each function, look at the final version of `udr.c` or view the changes function-by-function in the following table (the functions might not be in the same order as in your `udr.c` file).

Function Name	Description of Change
<code>CTempAssign()</code>	Verify that an entered CTemp value is greater than -273.15; send an error message if it is not.
<code>CTempCast()</code>	Verify that an entered FTemp value is greater than -459.6; perform conversion to Celsius if it is or send an error message if it is not.
<code>DecToTemp()</code>	Because of data type impedance between SQL and C, use the ESQL/C conversion function <code>dectodbl()</code> to convert the Informix server decimal data type to the C double type. The double precision data type is defined as the C type double .
<code>FTempAssign()</code>	Verify that an entered FTemp value is greater than -459.6; send an error message if it is not.
<code>FTempCast()</code>	Verify that an entered CTemp value is greater than -273.15; perform conversion to Fahrenheit if it is or send an error message if it is not.
<code>IntToTemp()</code>	Both integer and double precision are defined as C types, so you can make the conversion in C.

- Save the changes to `udr.c`.

Exercise 3: Creating Distinct Data Types and Casts, continued

11

Compile your source code using your C compiler.

See "[Compiling and Linking DataBlade Module Code](#)" for detailed instructions on using the Visual C++ makefile (**Mercury.mak** for this exercise) to compile the source code into the dynamic link library file (**Mercury.bld**).

For more information about compiling on UNIX, see the *DataBlade Developers Kit User's Guide*.

12

Install the Mercury DataBlade module.

✓ Install the Mercury DataBlade module in the `%INFORMIXDIR%\extend\Mercury.1.0` directory.

See "[Installing a DataBlade Module](#)" for detailed installation instructions.

✓ Ensure that the permissions on the `Mercury.bld` file are read-only. (If your dynamic link library or shared object file is not read-only, a database server error occurs when you try to load it.)

13

Start the database server.

See "[Starting Informix Dynamic Server on Windows NT](#)" for detailed instructions for starting Informix Dynamic Server on Windows NT.

For instructions on starting Informix Dynamic Server on UNIX, see the *Administrator's Guide* for your database server.

14

Create a test database.

✓ If the database in which you plan to test the Mercury DataBlade module does not yet exist, create it using the following SQL statement:

```
create database mercury_ex with buffered log;
```

15

Register the Mercury DataBlade module in the `mercury_ex` database.

See "[Registering a DataBlade Module](#)" for detailed instructions on registering using the BladeManager graphical user interface.

For information on registering using BladeManager from the command line, see the *DataBlade Module Installation and Registration Guide*.

16

Run the Mercury DataBlade module.

After you install and register the DataBlade module, the Informix database accepts the new data types and routines and runs them as if they were built-in.

To test the Mercury DataBlade module, execute [these SQL operations](#) using DB-Access or SQL Editor

1. Create the tables **climate_EU** and **climate_USA**.
2. Insert sample data into the two tables.
3. Run three queries against these tables.

Tutorial Exercise 4

Creating Row Data Types

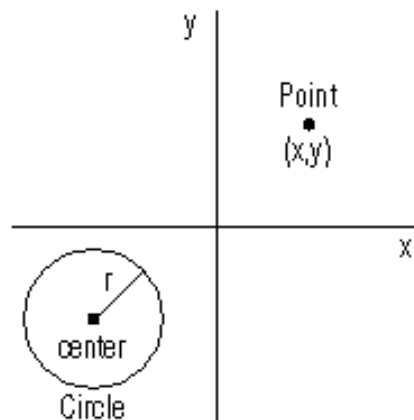
This exercise demonstrates creating [row data types](#) and user-defined routines that operate on them.

The RowCircle DataBlade module creates two new data types, **rciPoint** and **rciCircle**, and two new functions, **rciDistance()** and **rciContains()**.

The RowCircle DataBlade module enables you to:

- store circles or points in a single column of a database table.
- calculate the distance between two points.
- calculate whether a circle contains a point.

The following diagram illustrates the type of data RowCircle data types contain.



Tutorial Steps

This exercise consists of 13 steps. It takes approximately one hour to complete.

1

Create a new project.

- ✓ Start BladeSmith from the **Informix** program group and choose **Project→New**.

✓ Enter the following information into the New Project wizard.

Project Property	Value
DataBlade module name	RowCircle
Object prefix	rci
Description locale	en_us.1252
Server compatilby	9.13 or 9.14
Project version number (<i>Major.Minor</i>)	1.0
Project description	This DataBlade module supports the rciPoint and rciCircle row types.
Vendor unique ID (optional)	Your company's name
Company information	Your company's name, copyright, and contact information

✓ Click **Finish**.

✓ Choose **Project→Save**; create a new directory called **RowCircle** in any convenient place on your hard drive and save your project in this directory as a file called **RowCircle.ibs**.

Exercise 4: Creating Row Data Types, continued

2

Add the **rciPoint** data type.

✓ Choose **Edit→Insert→Row Type** and enter the following information in the New Row Type wizard.

Row Type Property	Value
Row type name	rciPoint
Inheritance	No parent row type
Columns	1. Name: x , type: double precision , cannot be NULL 2. Name: y , type: double precision , cannot be NULL (It does not make sense for this data type to allow a NULL coordinate.)
Grant execute privileges	Grant usage privileges

✓ Click **Finish** and then choose **Project→Save** to save your project.

3

Add the **rciDistance()** routine.

✓ Choose **Edit→Insert→Routine** and enter the following information in the New Routine wizard.

Routine Property	Value
Routine Name (SQL)	rciDistance
Return type	double precision
Statement Local Variable	No
Arguments	1. Name: point1 , Type: rciPoint , Default: none 2. Name: point2 , Type: rciPoint , Default: none
Language	C
Options for how the database server calls this routine (page 5)	Only the Routine does not accept NULL arguments check box should be checked; uncheck the Routine may return inconsistent check box.
Routine Name (C)	rciDistance
Shared object path	Leave as default
Behavior of routines	Well-behaved
Special stack requirements	None
Cost	0
Related routines	None
Grant execute privileges	Grant execute privileges

✓ Click **Finish** and then choose **Project→Save** to save your project.

Exercise 4: Creating Row Data Types, continued

4

Add the **rciCircle** data type.

✓ Choose **Edit→Insert→Row Type** and enter the following information in the New Row Type wizard.

Row Type Property	Value
Row type name	rciCircle
Inheritance	No parent row type
Columns	1. Name: center , type: rciPoint , cannot be NULL 2. Name: radius , type: double precision , cannot be NULL
Grant execute privileges	Grant usage privileges

✓ Click **Finish** and then choose **Project→Save** to save your project.

5

Add the **rciContains()** routine.

✓ Choose **Edit→Insert→Routine** and enter the following information in the New Routine wizard.

Routine Property	Value
Routine Name (SQL)	rciContains
Return type	Boolean
Statement Local Variable	No
Arguments	1. Name: circle , Type: rciCircle , Default: none 2. Name: point , Type: rciPoint , Default: none
Language	C
Options for how the database server calls this routine (page 5)	Only the Routine does not accept NULL arguments check box should be checked; uncheck the Routine may return inconsistent check box.
Routine Name (C)	rciContains
Shared object path	Leave as default
Behavior of routines	Well-behaved
Special stack requirements	None

Routine Property	Value
Cost	0
Related routines	None
Grant execute privileges	Grant execute privileges

✓ Click **Finish** and then choose **Project**→**Save** to save your project.

Exercise 4: Creating Row Data Types, continued

6

Generate the source code and the SQL registration scripts.

See "[Generating Code](#)" for detailed instructions.

BladeSmith generates the basic C code and SQL scripts necessary for your DataBlade module to run:

- The source code is saved in the **RowCircle\src\C** directory.
- The SQL scripts are saved in the **RowCircle\scripts** directory.

For more information about the **Generate DataBlade** command, see the *DataBlade Developers Kit User's Guide*.

The BladeSmith portion of this exercise is complete; you can exit BladeSmith.

7

Add source code to the templates generated by BladeSmith.

BladeSmith generates two source code files for the RowCircle DataBlade module:

- **udr.c**. For the user-defined routines, **rciDistance()** and **rciContains()**
- **support.c**. For tracing and BladeSmith utility functions (do not edit this file)

✓ Add the following statement to the **#include** statements at the top of the **udr.c** file to include the C math library:

```
#include <math.h>
```

Because the **rciContains()** function calls the **rciDistance()** function, you must have a definition of the **rciDistance()** function in the **udr.c** file before the code for the **rciContains()** function.

✓ Add the following **rciDistance()** function prototype directly after the **#include** statements in **udr.c**:

```
UDREXPOR mi_double_precision *rciDistance
(
  MI_ROW      * point1,
  MI_ROW      * point2,
  MI_FPARAM   * Gen_fparam
);
```

- ✓ Implement the `rciDistance()` function.

To implement the `rciDistance` function

1. Following the comment containing `Your_Code ... BEGIN`, find the comment that begins with `TO DO: Remove this comment`. Remove the comment and the `mi_db_error_raise()` statement and replace them with the following code:

```
{
    MI_DATUM x1, y1;
    MI_DATUM x2, y2;
    mi_integer retlen, retval;
    /* Fetch values from rows */
    retval = mi_value(point1, 0, &x1, &retlen);
    retval = mi_value(point1, 1, &y1, &retlen);
    retval = mi_value(point2, 0, &x2, &retlen);
    retval = mi_value(point2, 1, &y2, &retlen);
}
```

The elements of the `rciPoint` row data type, the *x*- and *y*-coordinates of a point, are extracted using the `mi_value()` statement and are stored in variables of type `MI_DATUM`. See the *DataBlade API Programmer's Manual* for information about handling data in row types.

2. Look at the `mi_alloc()` statement that follows your just-entered code. BladeSmith generates this `mi_alloc()` statement to take care of memory allocation for the function's return value, `Gen_RetVal`:

```
Gen_RetVal =
    (mi_double_precision *)mi_alloc( sizeof( mi_double_precision ) );
```

3. Find the comment `TO DO: Compute and store your value into Gen_RetVal`. Replace the statement `Gen_RetVal = 0;` with the following code:

```
*Gen_RetVal =
    sqrt( (*(mi_double_precision *)x1 - *(mi_double_precision *)x2) *
          (*(mi_double_precision *)x1 - *(mi_double_precision *)x2) +
          (*(mi_double_precision *)y1 - *(mi_double_precision *)y2) *
          (*(mi_double_precision *)y1 - *(mi_double_precision *)y2));
}
```

This statement calculates the distance between two points using the [Pythagorean formula](#).

The coordinates of each of the two points are cast from `MI_DATUM` to `mi_double_precision` values for the calculation.

✓ Implement the **rciContains()** function by replacing the code between the comment containing `Your_Code ... BEGIN` and the comment containing `Your_Code ... END` with the following code fragment:

```

{
    mi_double_precision * dist;
    mi_integer retlen;
    MI_DATUM center, radius;

    /*
    ** Fetch values from rows
    */
    mi_value( circle, 0, &center, &retlen );
    mi_value( circle, 1, &radius, &retlen );

    /*
    ** Computes the distance between
    ** the center of the circle and
    ** the point.
    */
    dist = rciDistance( point, center, Gen_fparam );

    /* Is the distance within the radius? */
    if (( *dist - *(mi_double_precision *)radius) <= 0)
    {
        Gen_RetVal = 1;
    }
    else
    {
        Gen_RetVal = 0;
    }
}

```

BladeSmith makes the return value an **mi_integer** type, which is the closest available C data type to the SQL Boolean return type you defined. This mismatch between the C and SQL type systems is known as *impedance*. The C code in user-defined routines must map the routine's arguments and return values between the two type systems.

The elements of the **rciCircle** row data type, the radius and center of a circle, are extracted using the **mi_value()** statement and are stored in variables of type `MI_DATUM`. See the *DataBlade API Programmer's Manual* for information about handling row types in C.

To determine whether the point is contained within the circle, the **rciContains()** function first uses the **rciDistance()** function to calculate the distance between the center of the circle and the point. Then the **rciContains()** function subtracts the radius from the distance. If the result is negative, the point is contained by the circle; if the result is positive, the point is outside the circle.

In the subtraction, the radius is cast from an `MI_DATUM` type to an **mi_double_precision** type.

- ✓ Check your source code against the final version of **udr.c**.
- ✓ Save the changes to **udr.c**.

Exercise 4: Creating Row Data Types, continued

8

Compile your source code with a C compiler.

See "[Compiling and Linking DataBlade Module Code](#)" for detailed instructions on using the Visual C++ makefile (**RowCircle.mak** for this exercise) to compile the source code into the dynamic link library file (**RowCircle.bld**).

For more information about compiling on UNIX, see the *DataBlade Developers Kit User's Guide*.

9

Install the RowCircle DataBlade module.

✓ Install the RowCircle DataBlade module into the `%INFORMIXDIR%\extend\RowCircle.1.0` directory on the server machine.

See "[Installing a DataBlade Module](#)" for detailed installation instructions.

✓ Ensure that the permissions on the **RowCircle.bld** file are read-only. (If your dynamic link library or shared object file is not read-only, a database server error occurs when you try to run it.)

10

Start the database server.

See "[Starting Informix Dynamic Server on Windows NT](#)" for detailed instructions for starting Informix Dynamic Server on Windows NT.

For instructions on starting Informix Dynamic Server on UNIX, see the *Administrator's Guide* for your database server.

11

Create a test database.

✓ If the database in which you plan to test the RowCircle DataBlade module does not yet exist, create it using the following SQL statement:

```
create database rowtest with buffered log;
```

12

Register the RowCircle DataBlade module in the **rowtest** database.

See "[Registering a DataBlade Module](#)" for detailed instructions on registering using the BladeManager graphical user interface.

For information on registering using BladeManager from the command line, see the *DataBlade Module Installation and Registration Guide*.

13

Run the RowCircle DataBlade module.

After you install and register the DataBlade module, the Informix database accepts the new data types and routines and runs them as if they were built-in.

To test the RowCircle DataBlade module, execute [these SQL operations](#) using DB-Access or SQL Editor

1. Create the table **tst_Point** that contains an **rciPoint** column.
2. Insert sample data into **tst_Point**.
3. Run a SELECT statement that calculates the distance of a point from the origin (0, 0) for each record.
4. Create the table **tst_RCirc**.
5. Insert sample data into **tst_RCirc**.
6. Run a SELECT statement that calculates if a circle contains the origin (0, 0) for each record.

Tutorial Exercise 5

Creating Opaque Data Types

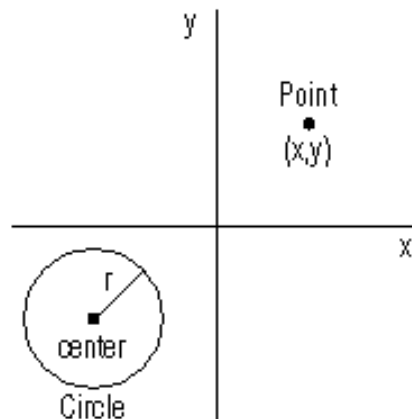
This exercise demonstrates creating [opaque data types](#), their [support routines](#), and user-defined routines that operate on them.

The Circle DataBlade module in this exercise is very similar to the RowCircle DataBlade module in Exercise 4. The difference is that the Circle DataBlade module creates [opaque data types](#), whereas the RowCircle DataBlade module creates [row data types](#).

The Circle DataBlade module creates two new data types, **Pnt** and **Circ**, and two new functions, **Distance()** and **Contains()**. Just like the RowCircle DataBlade module, the Circle DataBlade module enables you to:

- store circles or points in a single column of a database table.
- calculate the distance between two points.
- calculate whether a circle contains a point.

The following diagram illustrates the type of data Circle data types contain.



The Circle DataBlade module that you create in this exercise is similar to the example Circle DataBlade module included with the DataBlade Developers Kit. You can look at the example DataBlade module to see how to change the input format of an opaque circle type, and how to add additional functionality. The example Circle DataBlade module is located in the `%INFORMIXDIR%\dbdk\examples\types\dapi\Circle` directory.

Tutorial Steps

This exercise has 13 steps. It takes approximately one hour to complete.

1

Create the new project.

- ✓ Start BladeSmith from the **Informix** program group and choose **Project→New**.

✓ Enter the following information into the New Project wizard.

Project Property	Value
DataBlade module name	Circle
Object prefix	Leave blank
Description locale	en_us.1252
Server compatibilty	Your database server version
Project version number (<i>Major.Minor</i>)	1.0
Project description	This DataBlade module supports the Pnt and Circ opaque types.
Vendor unique ID (optional)	Your company's name
Company information	Your company's name, copyright, and contact information

✓ Click **Finish**.

✓ Choose **Project→Save**; create a new directory called **Circle** in any convenient place on your hard drive and save your project in this directory as a file called **Circle.ibs**.

Exercise 5: Creating Opaque Data Types, continued

2

Add the **Pnt** data type.

✓ Choose **Edit→Insert→Opaque Type** and enter the following information in the New Opaque Type wizard.

Opaque Type Property	Value
Opaque type name	Pnt
Server implementation	C
Client implemenatation	None
Define internal structure of opaque type to BladeSmith	Yes
Opaque type is fixed size	Yes

Opaque Type Property	Value
Internal members	1. Name: x , Type: mi_double_precision , Array: 0 2. Name: y , Type: mi_double_precision , Array: 0
Support routines	Uncheck these options: Text File Import/Export Binary File Import/Export Keep these options checked: Basic Text Input/Output Binary Send/Receive With Client Type Compare Support
Grant execute privileges	Grant usage privileges

✓ Click **Finish** and then choose **Project**→**Save** to save your project.

3

Add the **Circ** data type.

Choose **Edit**→**Insert**→**Opaque Type** and enter the following information in the New Opaque Type wizard.

Opaque Type Property	Value
Opaque type name	Circ
Server implementation	C
Client implemenatation	None
Define internal structure of opaque type to BladeSmith	Yes
Opaque type is fixed size	Yes
Internal members	1. Name: center , Type: Pnt , Array: 0 2. Name: radius , Type: mi_double_precision , Array: 0
Support routines	Uncheck these options: Text File Import/Export Binary File Import/Export Keep these options checked: Basic Text Input/Output Binary Send/Receive With Client Type Compare Support
Grant execute privileges	Grant usage privileges

✓ Click **Finish** and then choose **Project→Save** to save your project.

Exercise 5: Creating Opaque Data Types, continued

4

Add the **Distance()** routine.

✓ Choose **Edit→Insert→Routine** and enter the following information in the New Routine wizard.

Routine Property	Value
Routine Name (SQL)	Distance
Return type	double precision
Statement Local Variable	No
Arguments	1. Name: Pnt1 , Type: Pnt , Default: none 2. Name: Pnt2 , Type: Pnt , Default: none
Language	C
Options for how the database server calls this routine (page 5)	Only the Routine does not accept NULL arguments check box should be checked; uncheck the Routine may return inconsistent check box.
Routine Name (C)	Distance
Shared object path	Leave as default
Behavior of routines	Well-behaved
Special stack requirements	None
Cost	0
Related routines	None
Grant execute privileges	Grant execute privileges

✓ Click **Finish** and then choose **Project→Save** to save your project.

5

Add the **Contains()** routine.

✓ Choose **Edit→Insert→Routine** and enter the following information in the New Routine wizard.

Routine Property	Value
Routine Name (SQL)	Contains
Return type	Boolean
Statement Local Variable	No
Arguments	1. Name: Circ1 , Type: Circ , Default: none 2. Name: Pnt1 , Type: Pnt , Default: none
Language	C
Options for how the database server calls this routine (page 5)	Only the Routine does not accept NULL arguments check box should be checked; uncheck the Routine may return inconsistent check box.
Routine Name (C)	Contains
Shared object path	Leave as default
Behavior of routines	Well-behaved
Special stack requirements	None
Cost	0
Related routines	None
Grant execute privileges	Grant execute privileges

✓ Click **Finish** and then choose **Project→Save** to save your project.

Exercise 5: Creating Opaque Data Types, continued

6

Generate the source code and SQL registration scripts.

See "[Generating Code](#)" for detailed instructions.

BladeSmith generates the basic C code and SQL scripts necessary for your DataBlade module to run:

- The source code is saved in the **Circle\src\C** directory.
- The SQL scripts are saved in the **Circle\scripts** directory.

The BladeSmith portion of this exercise is complete; you can exit BladeSmith.

7

Add your source code to the templates generated by BladeSmith.

BladeSmith generates four source code files for the Circle DataBlade module:

- **Circ.c.** For the **Circ** opaque type support routines
- **Pnt.c.** For the **Pnt** opaque type support routines
- **udr.c.** For the user-defined routines, **Distance()** and **Contains()**
- **support.c.** For tracing and BladeSmith utility functions (do not edit this file)

✓ Look at the source code for the **Circ** and **Pnt** data type support routines in the **Circ.c** and **Pnt.c** files. You do not need to modify these files. BladeSmith creates complete support routines; the generated code manipulates the data types that compose the **Pnt** and **Circ** opaque types.

For each of the support routine types you selected in the New Opaque Type wizard, BladeSmith generates the following functions for the **Pnt** and **Circ** data types.

Support Routine Category	Routines for Pnt	Routines for Circ
Text Input and Output Functions	PntInput(), PntOutput()	CircInput(), CircOutput()
Binary Send and Receive Functions	PntSend(), PntReceive()	CircSend(), CircReceive()
Type Compare Functions	PntCompare(), PntEqual(), PntNotEqual()	CircCompare(), CircEqual(), CircNotEqual()

✓ Add the following statement to the **#include** statements at the beginning of the **udr.c** file to include the C math library:

```
#include <math.h>
```

Because the **rciContains()** function calls the **rciDistance()** function, you must have a definition of the **rciDistance()** function in the **udr.c** file before the code for the **rciContains()** function.

✓ Add the following **rciDistance()** function prototype directly after the **#include** statements in **udr.c**:

```
UDREXPOR mi_double_precision *Distance
(
Pnt      * Pnt1,
Pnt      * Pnt2,
MI_FPARAM * Gen_fparam
);
```

✓ Implement the **Distance()** function. This function implements the same functionality as the **rciDistance()** function in the RowCircle DataBlade module exercise.

To implement the Distance() function

1. Following the comment containing `Your_Code ... BEGIN`, find the comment below and remove it and the `mi_db_error_raise()` statement that follows it:

```

/*
** TO DO: Remove this comment and call to
** mi_db_error_raise after implementing
** this function.
*/
mi_db_error_raise( Gen_Con, MI_EXCEPTION,
    "Function Distance has not been implemented." );

```

2. Find the comment `TO DO: Compute and store your value into Gen_RetVal`. Replace the statement `Gen_RetVal = 0;` with the following code:

```

*Gen_RetVal =
    sqrt((Pnt1->x - Pnt2->x) * (Pnt1->x - Pnt2->x) +
        (Pnt1->y - Pnt2->y) * (Pnt1->y - Pnt2->y) );

```

This statement calculates the distance between two points using the [Pythagorean formula](#).

✓ Implement the **Contains()** function. This function implements the same functionality as the **rciContains()** function in the RowCircle DataBlade module exercise.

To implement the Contains() function

1. Between the comments `/* Your_Declarations ... BEGIN */` and `/* Your_Declarations ... END */`, add the following declaration:

```
double * dist;
```

2. Replace the code between the comment `/* Your_Code ... BEGIN*/` and the comment `/*Your_Code ... END*/` with the following code:

```

/*
** Computes the distance between
** the center of the circle and
** the point.
*/
dist = Distance( Pnt1, &Circl->center, Gen_fparam );

/* Is the distance within the radius? */
if( (*dist - Circl->radius) <= 0 )
{
    Gen_RetVal = 1;
}
else
{
    Gen_RetVal = 0;
}

```

BladeSmith makes the return value of **Contains()** an **mi_integer**, which is the closest available C data type to the SQL Boolean return type you defined.

To determine whether the point is contained within the circle, the **Contains()** function first uses the **Distance()** function to calculate the distance between the center of the circle and the point. Then the **Contains()** function subtracts the radius from the distance. If the result is negative, the point is contained by the circle; if the result is positive, the point is outside the circle.

- ✓ Check your source code against the final version of **udr.c**.
- ✓ Save the changes to **udr.c**.

Exercise 5: Creating Opaque Data Types, continued

8

Compile the source code with a C compiler.

See "[Compiling and Linking DataBlade Module Code](#)" for detailed instructions on using the Visual C++ makefile (**Circle.mak** for this exercise) to compile the source code into the dynamic link library file (**Circle.bld**).

For more information about compiling on UNIX, see the *DataBlade Developers Kit User's Guide*.

9

Install the Circle DataBlade module.

- ✓ Install the Circle DataBlade module into the **%INFORMIXDIR%\extend\Circle.1.0** directory on the server machine.

See "[Installing a DataBlade Module](#)" for detailed installation instructions.

- ✓ Ensure that the permissions on the **Circle.bld** file are read-only. (If your dynamic link library or shared object file is not read-only, a database server error occurs when you try to run it.)

10

Start the database server.

See "[Starting Informix Dynamic Server on Windows NT](#)" for detailed instructions for starting Informix Dynamic Server on Windows NT.

For instructions on starting Informix Dynamic Server on UNIX, see the *Administrator's Guide* for your database server.

11

Create a test database.

If the database in which you plan to test the Circle DataBlade module does not yet exist, create it using the following SQL statement:

```
create database opaquetest with buffered log;
```

12

Register the Circle DataBlade module in the **opaquetest** database.

See "[Registering a DataBlade Module](#)" for detailed instructions on registering using the BladeManager graphical user interface.

For information on registering using BladeManager from the command line, see the *DataBlade Module Installation and Registration Guide*.

13

Run the Circle DataBlade module.

After you install and register the Circle DataBlade module, the Informix database server accepts the new data types and routines and runs them as if they were built-in.

To test the Circle DataBlade module, execute [these SQL operations](#) using DB-Access or SQL Editor

1. Create a table **tst_Pnt** that contains a **Pnt** column.
2. Insert sample data into **tst_Pnt**.
3. Run a SELECT statement that calculates the distance of a point from the origin (0, 0) for each record.
4. Create a table **tst_Circ** that contains a **Circ** column.
5. Insert sample data into **tst_Circ**.
6. Run a SELECT statement that calculates if a circle contains the origin (0, 0) for each record.

Tutorial Exercise 6

Using Interfaces

This exercise demonstrates using interfaces to define how one DataBlade module depends on another.

You will create two DataBlade modules, Point and Circle:

- The Point DataBlade module contains the **Point** opaque data type and the **Distance()** function.
- The Circle DataBlade module contains the **Circle** opaque data type and the **Contains()** function.

Because the **Circle** data type has a member of type **Point**, and the **Contains()** function uses the **Distance()** function, the Circle DataBlade module depends on the Point DataBlade module. To use the Circle DataBlade module, users must first install and register the Point DataBlade module.

By using interfaces, you set up a dependency so users cannot register the Circle DataBlade module unless the Point DataBlade module is already registered. If a user tries to register the Circle DataBlade module first, BladeManager looks for the Point DataBlade module and performs one of the following actions:

- If the Point DataBlade module is already installed on the database server, BladeManager prompts the user to register it, and only then proceeds to register the Circle DataBlade module.
- If the Point DataBlade module is not installed, BladeManager does not register the Circle DataBlade module.

This exercise also shows how a DataBlade module routine can call a routine from another DataBlade module directly using a [wrapper function](#) instead of an SQL statement.

Tutorial Steps

This exercise has 18 steps. It takes approximately one hour to complete.

1

Create the Point DataBlade module.

- ✓ Start BladeSmith from the Informix program group and choose **Project→New**.
- ✓ Enter **Point** for the DataBlade module name, specify your database server version, leave all other fields in the wizard as their default values, and then click **Finish**.
- ✓ Choose **Project→Save**; create a new directory called **Point** in any convenient place on your hard drive and save your project in this directory as a file called **Point.ibs**.

2

Add the **Point** data type.

✓ Choose **Edit→Insert→Opaque Type**, enter the following information in the New Opaque Type wizard, and leave all other fields in the wizard as their default values.

Opaque Type Property	Value
Opaque type name	Point
Internal members	1. Name: x , Type: mi_double_precision , Array: 0 2. Name: y , Type: mi_double_precision , Array: 0

✓ Click **Finish** and then choose **Project→Save** to save your project.

3

Add the **Distance()** routine to the Point DataBlade module.

✓ Choose **Edit→Insert→Routine**, enter the following information in the New Routine wizard, and leave all other fields in the wizard as their default value.

Routine Property	Value
Routine name (SQL)	Distance
Return type	double-precision
Arguments	1. Name: Pnt1 , type: Point 2. Name: Pnt2 , type: Point

✓ Click **Finish** and then choose **Project→Save** to save your project.

4

Add the Point interface to the Point DataBlade module.

✓ Choose **Edit→Insert→Interface**, enter `IPoint` for the Point interface name, and enter a description, such as “The Point interface supports the opaque type Point and the Distance() routine.”

✓ Click **Finish** and then choose **Project→Save** to save your project.

Exercise 6: Using Interfaces, continued

5

Create the project for the Circle DataBlade module.

- ✓ Choose **Project→New**.
- ✓ Enter **Circle** for the DataBlade module name, specify your database server version, leave all other fields in the wizard as their default values; and then click **Finish**.
- ✓ Choose **Project→Save**; create a new directory called **Circle** in any convenient place on your hard drive and save your project in this directory as a file called **Circle.ibs**. Be sure to save this project in a different directory than the Circle DataBlade module you created in Exercise 5.

6

Import the **Point** data type and **IPoint** interface definitions into the Circle DataBlade module project.

Because the **Circle** data type contains an element of **Point** type, you must have the definition of a **Point** type available within the Circle DataBlade module project.

To import the Point data type

1. Open both the Point DataBlade module project and the Circle DataBlade module project.
2. Click the Point DataBlade module project window to make it active.
3. In the folder **User Defined Objects\Types\Opaque Types**, select the **Point** data type.
4. Choose **Edit→Copy**.
5. Click the Circle DataBlade module project window to make it active.
6. Choose **Edit→Import→From Clipboard**. BladeSmith copies the required references for the **Point** data type to the **Imported Objects** folder of the Circle DataBlade project.

You must also include the definition of the **IPoint** interface in the Circle DataBlade module project. The interface informs BladeManager that the Point Datablade module is required by the Circle Datablade module.

To import the IPoint interface

1. Click the Point DataBlade module project window to make it active.
 2. In the folder **User Defined Objects\Interfaces**, select the **IPoint** interface.
 3. Choose **Edit→Copy**.
 4. Click the Circle DataBlade module project window to make it active.
 5. Choose **Edit→Import→From Clipboard**. BladeSmith copies the required references for the **IPoint** interface to the **Imported Objects** folder of the Circle DataBlade module project.
- ✓ Click **Finish** and then choose **Project→Save** to save your project.

7

Add the **Circle** data type to the Circle DataBlade module.

✓ Choose **Edit→Insert→Opaque Type**, enter the following information in the New Opaque Type wizard, and leave all other fields in the wizard as their default values.

Opaque Type Property	Value
Opaque type name	Circle
Internal members	1. Name: center , Type: Point , Array: 0 2. Name: radius , Type: mi_double_precision , Array: 0

✓ Click **Finish** and then choose **Project→Save** to save your project.

8

Add the **Contains()** routine to the Circle DataBlade module.

✓ Choose **Edit→Insert→Routine**, enter the following information in the New Routine wizard, and leave all other fields in the wizard as their default values.

Routine Property	Value
Routine name (SQL)	Contains
Return type	Boolean
Arguments	1. Name: Circ1 , type: Circle 2. Name: Pnt1 , type: Point

✓ Click **Finish** and then choose **Project→Save** to save your project.

9

Generate the source code for both the Point and Circle DataBlade modules.

See "[Generating Code](#)" for detailed instructions.

BladeSmith generates the basic C code and SQL scripts necessary for your DataBlade modules to run:

- The source code is saved in the **Point\src\C** and **Circle\src\C** directories.
- The SQL scripts are saved in the **Point\scripts** and **Circle\scripts** directories.

The BladeSmith portion of this exercise is complete; you can exit BladeSmith.

Exercise 6: Using Interfaces, continued

10

Add source code to the Point DataBlade module **udr.c** file.

You must add code to implement the **Distance()** function to the **udr.c** file in the **Point\src\C** directory. This function implements the same functionality as the **rciDistance()** function in Exercise 4 and the **Distance()** function in Exercise 5.

✓ Add the following statement to the **#include** statements at the beginning of the **udr.c** file to include the C math library:

```
#include <math.h>
```

✓ Following the comment containing **Your_Code ... BEGIN**, find the comment below and remove it and the **mi_db_error_raise()** statement that follows it:

```
/*
** TO DO: Remove this comment and call to *
*       mi_db_error_raise after implementing
**       this function.
*/
mi_db_error_raise( Gen_Con, MI_EXCEPTION,
                  "Function Distance has not been implemented." );
```

✓ Find the comment **TO DO: Compute and store your value into GenRetVal**. Replace the statement **GenRetVal = 0;** with the following code:

```
*GenRetVal =
    sqrt((Pnt1->x - Pnt2->x) * (Pnt1->x - Pnt2->x) +
         (Pnt1->y - Pnt2->y) * (Pnt1->y - Pnt2->y) );
```

✓ Check your source code against the final version of **udr.c**.

✓ Save the changes to **udr.c**.

11

Add source code to the Circle DataBlade module **udr.c** file.

Add code to implement the **Contains()** function to the **udr.c** file in the **Circle\src\C** directory. This function implements the same functionality as the **rciContains()** function in Exercise 4 and the **Contains()** function in Exercise 5.

Because the **Contains()** function in the Circle DataBlade module calls the **Distance()** function in the Point DataBlade module, you must include a **wrapper function** for the **Distance()** function in the Circle DataBlade module **udr.c** file. The **Contains()** function calls the wrapper function, which loads the **Distance()** function from the Point DataBlade module, executes it, and returns the result to the **Contains()** function.

- ✓ Add the following wrapper function to the **udr.c** file for the Circle DataBlade module after the list of **#include** statements:

```
mi_double_precision *Distance
(
MI_CONNECTION * Gen_Con,
Point * Pnt1,
Point * Pnt2
)
{
MI_FUNC_DESC * fd = NULL;
MI_DATUM ret = NULL;
mi_integer error = 0;

fd = mi_routine_get( Gen_Con, 0, "function Distance(Point, Point)");
if (fd == NULL)
    DBDK_TRACE_ERROR( "Contains",
        "function Distance(Point, Point) not gotten.", 10 );

ret = mi_routine_exec( Gen_Con, fd, &error, Pnt1, Pnt2);
if (error != MI_OK)
    DBDK_TRACE_ERROR( "Contains",
        "exec function Distance(Point, Point) failed.", 10 );

return ret;
}
```

- ✓ Replace the statement **Gen_Con = NULL;** with the following connection statement:

```
Gen_Con = mi_open( NULL, NULL, NULL );
/* Verify that the connection has been established. */
if( Gen_Con == 0 )
{
    /*
    ** Opening the current connection has failed
    ** so issue the following message and quit.
    **
    **      "Connection has failed in Contains."
    */
    DBDK_TRACE_ERROR( "Contains", ERRORMESG1, 10 );

    /* not reached */
}
```

- ✓ Replace the code between the comment `/* Your_Code ... BEGIN*/` and the comment `/*Your_Code ... END*/` with the following code:

```
/*
** Computes the distance between
** the center of the circle and
** the point.
*/
{
    double * dist = Distance( Gen_Con, Pnt1, &Circl->center );

    /* Is the distance within the radius? */
    if( (*dist - Circl->radius) <= 0 )
    {
        Gen_RetVal = 1;
    }
    else
    {
        Gen_RetVal = 0;
    }
}
```

- ✓ Check your source code against the final version of **udr.c**.
- ✓ Save the changes to **udr.c**.

Exercise 6: Using Interfaces, continued

12

Compile the source code for both DataBlade modules with a C compiler.

See "[Compiling and Linking DataBlade Module Code](#)" for detailed instructions on using the Visual C++ makefile (**Point.mak** and **Circle.mak** for this exercise) to compile the source code into the dynamic link library file (**Point.bld** and **Circle.bld**).

For information about compiling on UNIX, see the *DataBlade Developers Kit User's Guide*.

13

Install the Point DataBlade module.

- ✓ Install the Point DataBlade module into the **%INFORMIXDIR%\extend\Point.1.0** directory on the database server machine.

See "[Installing a DataBlade Module](#)" for detailed installation instructions.

✓ Ensure the permissions on the **Point.bld** file are read-only. (If your DLL or shared object file is not read-only, a server error occurs when you try to run it.)

14

Install the Circle DataBlade module.

✓ Install the Circle DataBlade module into the `%INFORMIXDIR%\extend\Circle.2.0` directory on the server machine. (If you install this DataBlade module in the `%INFORMIXDIR%\extend\Circle.1.0` directory, you overwrite the Circle DataBlade module you created in Exercise 5.)

See "[Installing a DataBlade Module](#)" for detailed installation instructions.

✓ Ensure the permissions on the **Circle.bld** file are read-only. (If your DLL or shared object file is not read-only, a server error occurs when you try to run it.)

15

Start the database server.

See "[Starting Informix Dynamic Server on Windows NT](#)" for detailed instructions for starting Informix Dynamic Server on Windows NT.

For instructions on starting Informix Dynamic Server on UNIX, see the *Administrator's Guide* for your database server.

16

Create a test database.

If the database where you plan to test the Circle and Point DataBlade modules does not yet exist, create it using the following SQL statement:

```
create database intftest with buffered log;
```

17

Register the Point and Circle DataBlade modules in the database.

Because you have defined the Circle DataBlade module as depending on the Point DataBlade module, BladeManager does not let you register the Circle DataBlade module unless the Point DataBlade module is already registered in that database.

To register the Point and Circle DataBlade modules

1. Start the BladeManager GUI from the **Informix** program group (you should be logged on as the default user for your workstation so that your NT or UNIX login is the same as your Informix user ID).

2. In the **Databases** page of the BladeManager window, click the name of the Informix database where you want to register your DataBlade modules.
3. From the **Available** list, select **Circle.2.0** and click **Add**; then click **Apply**.
4. The Modules with Missing Interfaces dialog box appears and lists the DataBlade modules available on your database server.
5. Select the Point DataBlade module and click **OK**. BladeManager registers both the Circle and the Point DataBlade modules in the database.
6. Click **Exit** to close BladeManager.

18

Run the Point and Circle DataBlade modules.

You can test the success of the Point interface by testing the Circle DataBlade module.

To test the Circle DataBlade module, execute [these SQL operations](#) using DB-Access or SQL Editor

1. Create the table **tst_Circle**.
2. Insert sample data into **tst_Circle**.
3. Run a SELECT statement that calculates if a circle contains the origin (0, 0) for each record.

Tutorial Exercise 7

Using Smart Large Objects

This exercise demonstrates creating a DataBlade module that handles [smart large objects](#).

Smart large objects allow you to store large data files in a way similar to that used by an operating system file system, while controlling, accessing, and altering files with the database server.

A smart large object has two parts:

- The data file, which is stored in an [sbspace](#)
- The handle, which is stored in the database and contains the location of the data file in the sbspace

For more information on smart large objects, see the *DataBlade API Programmer's Manual*.

The Poem DataBlade module creates one new opaque data type, **Poem**, and the internal support routines the database server uses to process the **Poem** data type. The **Poem** data type contains a single member: a handle to a smart large object. The data for the **Poem** data type are XML files, each containing a poem.

Tutorial Steps

This exercise has 11 steps. It takes approximately one hour to complete.

1

Create a new project.

- ✓ Start BladeSmith from the **Informix** program group and choose **Project→New**.
- ✓ Enter `Poem` for the DataBlade module name, specify your database server version, leave all other fields in the wizard as their default values, and then click **Finish**.
- ✓ Choose **Project→Save**; create a new directory called **Poem** in any convenient place on your hard drive and save your project in this directory as a file called **Poem.ibs**.

2

Add the **Poem** data type.

- ✓ Choose **Edit→Insert→Opaque Type**, enter the following information in the New Opaque Type wizard, and leave all other fields in the wizard as their default values.

Opaque Type Property	Value
Opaque type name	Poem
Internal members	Name: poem , Type: <code>MI_LO_HANDLE</code> , Array: 0

- ✓ Click **Finish** and then choose **Project→Save** to save your project.

3

Add custom SQL to test for an sbspace.

Because you are creating smart large objects, you must have a default sbspace defined in your database server. If you do not have an sbspace, BladeManager does not register your DataBlade module.

BladeManager can execute a routine, **SYSBldTstSBSpace()**, during registration to check for the required sbspace. If the sbspace is found, BladeManager successfully registers the DataBlade module. If the sbspace is not found, BladeManager does not register the DataBlade module, and an error appears in the registration log explaining the problem. You instruct BladeManager to run **SYSBldTstSBSpace()** by adding a [custom SQL file](#) to your DataBlade module project.

✓ Choose **Edit→Insert→SQL File**, enter the following information in the New SQL File wizard, and leave all other fields in the wizard as their default values.

Custom SQL Property	Value
Descriptive name for SQL	SbspaceTest
Custom SQL create text	EXECUTE FUNCTION SYSBldTstSBSpace(" ");
These objects require this SQL	Poem

✓ Click **Finish** and then choose **Project→Save** to save your project.

4

Generate the source code and SQL registration scripts.

See "[Generating Code](#)" for detailed instructions.

BladeSmith generates the basic C code and SQL scripts necessary for your DataBlade module to run:

- The source code is saved in the **Poem\src\C** directory.
- The SQL scripts are saved in the **Poem\scripts** directory.

The BladeSmith portion of this exercise is complete; you can exit BladeSmith.

Exercise 7: Using Smart Large Objects, Continued

5

Modify source code.

The source code for the Poem DataBlade module is in the **Poem.c** file in the **src/c** directory.

The following table lists the internal support functions for the **Poem** opaque data type.

Support Routine Category	Smart Large Object Handling
Text Input and Output Functions	<p>The PoemInput() function creates a smart large object handle and inserts it in the database table column. Then the function assigns a location in the sbspace for the data file and copies the file to that location.</p> <p>The PoemOutput() function reads the smart large object handle and generates a random filename for the data file. Then the function creates a file on the client computer and copies the smart large object data into it.</p>
Binary Send and Receive Functions	The PoemSend() and PoemReceive() functions move the binary format of the smart large object handle between the database server and client computers.
Binary File Import and Export Functions	The PoemImportBinary() and PoemExportBinary() functions perform bulk copies of the binary format of the smart large object handle between the database server and external files.
Text File Import and Export Functions	The PoemImportText() and PoemExportText() functions perform bulk copies of the text format of the smart large object handle between the database server and external files.
Contains Large Objects Routines	<p>The PoemLOhandles() function returns a list of the MI_LO_HANDLES structures in an opaque data type. The database server calls this function when archiving or performing oncheck operations.</p> <p>The PoemAssign() function adds to the reference count for the smart large object before it saves the handle in the database and the data in the sbspace (if it is not already present).</p> <p>The PoemDestroy() procedure decrements the reference count for the smart large object before deleting the handle from the database and the data from the sbspace (if the reference count is zero).</p>
Type Compare Functions	<p>The PoemCompare() function compares the handles of two opaque data types and determines whether they are the same.</p> <p>The PoemEqual() and PoemNotEqual() functions call the PoemCompare() function.</p>

The default behavior for the **PoemOutput()** function is to generate a random number filename. However, you can modify the source code for this function so that it generates a meaningful filename: the title of the poem. The data files for the Poem DataBlade module are XML files containing poems by Edgar Allan Poe. The title of the poem appears in the beginning of the file; you can extract the title and use it to create the filename. You use DataBlade API functions to open and read smart large objects; the process is similar to opening and reading a file.

✓ In the **PoemOutput()** function, add the following code after the `/* Format the attribute value into the output string. */` comment:

```
{
char PoemStart[128];
char * pTag1 = NULL;
char * pTag2 = NULL;
MI_LO_FD LO_fd;

/*
 * Get the poem title: open the smart large object, read the first 127 bytes,
 * look for the begin-title and end-title XML tags, then close the object.
 */
LO_fd = mi_lo_open( Gen_Con, &Gen_InData->poem, MI_LO_RANDOM | MI_LO_RDONLY );
if (LO_fd != MI_ERROR)
    {
    mi_lo_read( Gen_Con, LO_fd, PoemStart, 127);
    PoemStart[127] = 0;
    pTag1 = strstr( PoemStart, "<title>" );
    pTag2 = strstr( PoemStart, "</title>" );
    if (pTag2 != NULL)
        *pTag2 = 0;
    mi_lo_close( Gen_Con, LO_fd );
    }

/*
 * Build the filename: if the title tags are found, use the poem title plus an
 * ".xml" extension as the filename, otherwise, use the name "poem.xml".
 */
if (pTag1 != NULL)
    {
    strcpy( Gen_LOFile, pTag1 + 7 );
    strcat(Gen_LOFile, ".xml!" );
    /* The "!" indicates that the filename need not be unique. */
    }
else
    strcpy( Gen_LOFile, "poem.xml" );

}
```

For more information on the **mi_lo_open()**, **mi_lo_read()**, and **mi_lo_close()** functions, see the *DataBlade API Programmer's Manual*. The **Gen_LOFile()** function is a BladeSmith utility function that calls the **mi_lo_to_file()** DataBlade API function. For information on the **mi_lo_to_file()** function and the ! filename wildcard symbol, see the *DataBlade API Programmer's Manual*.

- ✓ Remove the following statement after the `/* Save the large object to disk. */` comment:

```
strcpy( Gen_LOFile, LO_FN_MASK );
```

- ✓ Check your source code against the final version of **Poem.c**.
- ✓ Save the changes to **Poem.c**.

Exercise 7: Using Smart Large Objects, Continued

6

Compile the source code with a C compiler.

See "[Compiling and Linking DataBlade Module Code](#)" for detailed instructions on using the Visual C++ makefile (**Poem.mak** for this exercise) to compile the source code into the dynamic link library file (**Poem.bld**).

For more information about compiling on UNIX, see the *DataBlade Developers Kit User's Guide*.

7

Install the Poem DataBlade module.

- ✓ Install the Poem DataBlade module into the `%INFORMIXDIR%\extend\Poem.1.0` directory on the server machine.

See "[Installing a DataBlade Module](#)" for detailed installation instructions.

- ✓ Ensure that the permissions on the **Poem.bld** file are read-only. (If your dynamic link library or shared object file is not read-only, a database server error occurs when you try to run it.)

8

Start the database server.

See "[Starting Informix Dynamic Server on Windows NT](#)" for detailed instructions for starting Informix Dynamic Server on Windows NT.

For instructions on starting Informix Dynamic Server on UNIX, see the *Administrator's Guide* for your database server.

9

Create a test database.

If the database in which you plan to test the Poem DataBlade module does not yet exist, create it using the following SQL statement:

```
create database smartlotest with buffered log;
```

10

Register the Poem DataBlade module in the **smartlotest** database.

See "[Registering a DataBlade Module](#)" for detailed instructions on registering using the BladeManager graphical user interface.

If registration fails, see "[Registration Failure with an sbspace Test](#)" for instructions.

For information on registering using BladeManager from the command line, see the *DataBlade Module Installation and Registration Guide*.

11

Run the Poem DataBlade module.

After you install and register the Poem DataBlade module, the Informix database server accepts the new data type and routines and runs them as if they were built-in.

When you select smart large objects from the database, the database server copies them to the current directory for your SQL query application (SQL Editor or DB-Access). The current directory depends on how the application is started:

- If you start the application from the **Informix** program group, the current directory is the **%INFORMIXDIR%\bin** directory.
 - If you start the application by double-clicking an SQL file, the current directory is the location of that SQL file.
- ✓ Copy the following files from the **%INFORMIXDIR%\dbdk\InfoShelf\tutorial\source** directory to the Poem DataBlade module project directory:
- **dorado.xml**
 - **forannie.xml**
 - **annabel.xml**
 - **poem.sql**
- ✓ Double-click the **poem.sql** file to open SQL Editor or DB-Access.

To test the Poem DataBlade module

1. Create the table.
2. Edit the SQL INSERT statements to reflect the directory to which you copied the XML files.
3. Insert three smart large objects into the database.
4. Retrieve the smart large objects; the database server copies them to the same directory as the original files but gives them different names.
5. Look at the retrieved XML files (named **Annabel Lee.xml**, **Eldorado.xml**, and **For Annie.xml**) using Notepad.