SYMANTEC.™

Visual Cafe™ Database Developer's Guide

Symantec Visual Cafe™ Database Developer's Guide

Copyright Notice

Trademarks

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

SYMANTEC LICENSE AND WARRANTY

The software which accompanies this license (the "Software") is the property of Symantec or its licensors and is protected by copyright law. While Symantec continues to own the Software, you will have certain rights to use the Software after your acceptance of this license. Except as may be modified by a license addendum which accompanies this license, your rights and obligations with respect to the use of this Software are as follows:

• You may:

(i) use one copy of the Software on a single computer;

(ii) make one copy of the Software for archival purposes, or copy the software onto the hard disk of your computer and retain the original for archival purposes;

(iii) use the Software on a network, provided that you have a licensed copy of the Software for each computer that can access the Software over that network;

(iv) after written notice to Symantec, transfer the Software on a permanent basis to another person or entity, provided that you retain no copies of the Software and the transferee agrees to the terms of this agreement; and

(v) if a single person uses the computer on which the Software is installed at least 80% of the time, then after returning the completed product registration card which accompanies the Software, that person may also use the Software on a single home computer.

• You may not:

(i) copy the documentation which accompanies the Software;

(ii) sublicense, rent or lease any portion of the Software;

(iii) reverse engineer, decompile, disassemble, modify, translate, make any attempt to discover the source code of the Software, or create derivative works from the Software; or

(iv) use a previous version or copy of the Software after you have received a disk replacement set or an upgraded version as a replacement of the prior version, unless you donate a previous version of an upgraded version to a charity of your choice, and such charity agrees in writing that it will be the sole end user of the product, and that it will abide by the terms of this agreement. Unless you so donate a previous version of an upgraded version, upon upgrading the Software, all copies of the prior version must be destroyed.

• Sixty Day Money Back Guarantee:

If you are the original licensee of this copy of the Software and are dissatisfied with it for any reason, you may return the complete product, together with your receipt, to Symantec or an authorized dealer, postage prepaid, for a full refund at any time during the sixty day period following the delivery to you of the Software.

• Limited Warranty:

Symantec warrants that the media on which the Software is distributed will be free from defects for a period of sixty (60) days from the date of delivery of the Software to you. Your sole remedy in the event of a breach of this warranty will be that Symantec will, at its option, replace any defective media returned to Symantec within the warranty period or refund the money you paid for the Software. Symantec does not warrant that the Software will meet your requirements or that operation of the Software will be uninterrupted or that the Software will be error-free.

THE ABOVE WARRANTY IS EXCLUSIVE AND IN LIEU OF ALL OTHER WARRANTIES, WHETHER EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHER RIGHTS, WHICH VARY FROM STATE TO STATE.

• Disclaimer of Damages:

REGARDLESS OF WHETHER ANY REMEDY SET FORTH HEREIN FAILS OF ITS ESSENTIAL PURPOSE, IN NO EVENT WILL SYMANTEC BE LIABLE TO YOU FOR ANY SPECIAL, CONSEQUENTIAL, INDIRECT OR SIMILAR DAMAGES, INCLUDING ANY LOST PROFITS OR LOST DATA ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE EVEN IF SYMANTEC HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

IN NO CASE SHALL SYMANTEC'S LIABILITY EXCEED THE PURCHASE PRICE FOR THE SOFTWARE. The disclaimers and limitations set forth above will apply regardless of whether you accept the Software.

• U.S. Government Restricted Rights:

RESTRICTED RIGHTS LEGEND. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c) (1) and (2) of the Commercial Computer Software-Restricted Rights clause at 48 CFR 52.227-19, as applicable, Symantec Corporation, 10201 Torre Avenue, Cupertino, CA 95014.

Language Addendum

If the Software is a Symantec language product, then you have a royalty-free right to include object code derived from the Symantec component (java source or class) files in programs that you develop using the Software and you also have the right to use, distribute, and license such programs to third parties without payment of any further license fees, so long as a copyright notice sufficient to protect your copyright in the program is included in the graphic display of your program and on the labels affixed to the media on which your program is distributed. You have the right to make changes to the Symantec components, but only to the extent necessary to correct bugs in such components, and not for any other purpose. You also have a royalty-free right to include unmodified (except as stated in the previous sentence) Symantec component files required by your programs, but not as components of any development environment or component library you are distributing. The Symantec component files that may be redistributed are in the following folder in the Visual Cafe directory - VisualCafe\redist. The Java Virtual Machine (VM) or Just In Time (JIT) compiler may not be redistributed.

Acknowledgements

Special thanks to Puru Balakrishnan, Vrinda Bangari, Ken Bradley, Cheryl Chambers, Carlos Chang, Ken Chizinsky, Orlando Cordero, Adrian Diaconescu, Jim Douglas, Dennis Dunham, Erfert Fenton, Asha Giridharan, Guy Haas, Kim Haeger, Steve Howard, Ryohei Ihara, Lauren Katzive, Geoffrey Leong, Landon Ott, Cheryl Potter, Vasudev Rao, Anand Ravipati, Michael Richion, Aparna Srikanth, Jonathon Simonoff, Robert Sumberac, Wani Tang, Glenn Taylor, Hristo Tonev, Alessandro Vernet, Rama Vijayakumar. Additional thanks to Derek Eclavea.

C O N T E N T S

Chapter 2 Developing a Databound Project

Chapter 3 Data Binding

Chapter 4 Managing Data and Connections

What's New in Visual Cafe Database Edition

This chapter describes the many exciting new features in Visual Cafe 3.0, Database Edition. This document is also available online from the Windows Start menu.

New components

Visual Cafe Database Edition 3.0 includes improved support for the JFC/ Swing model architecture. New databound components have been added to enhance your programs and reduce your development efforts to accomplish common and robust tasks. A databound component is an AWT or Swing component that can work with live data, such as data from a database. Many of these databound components contain their own customizers, editors, and wizards.

Swing databound components

In addition to the standard Swing (JFC) components that Visual Cafe offers, you can also use the Swing DataBound components to work with a variety of datasources. Visual Cafe also provides the mechanisms to help you bind Swing DataBound components with datasources.

For more information on DataBound Swing components, see "Datasource components" on page 4-13.

AWT DataBound components

AWT DataBound components are extended AWT components with added database functionality. You won't have to explicitly bind these components to datasources with one of the binding model components. For more information, see "AWT DataBound components" on page 1-6.

Datasource components

Datasource components provide and manage data between your program's users, the databound objects in a project, and the database. Datasource components also allow you to add stored procedure and calculated data support to your projects. Wizards and customizers that help you add and customize datasource components are provided to ensure rapid development of complex programs. For more information, see "Data Source components" on page 1-9.

Stored procedure support

With Visual Cafe Database Edition, you can add support for database stored procedures to your project. Visual Cafe provides adapters that bring the functionality of stored procedures to your project and to the users of your programs. For more information, see "Working with stored procedures" on page 3-67.

Validation support

You use validation to control data integrity in your program and to issue server side calls for validating user input. Then, you can automatically or programatically notify the user when the entered data is not within specified parameters. You can also use the `Validation Adapter` in conjunction with other adapters, such as the `Calculation Adapter`, to ensure that the correct type of data is passed to the objects requesting it. For more information, see "Calculation and validation rules" on page 3-52

Calculation support

Use the `Calculation Adapter` to implement some of the most common calculation rules that you want to include in a project. The predefined library of numeric and string rules include addition, subtraction, multiplication, division, and summation. You can create and import your

own rules to add complex calcuations and to spawn server-side logic. For more information, see "Calculation and validation rules" on page 3-52.

Data Connection Navigator

The Data Connection Navigator (dcNAVIGATOR) lets you navigate and add datasources quickly to your project. The dcNAVIGATOR is accessible from wizards and editors where you need to specify or change a datasource. It presents you with a hierarchical view of datasources currently in your program. For more information, see "Using the Data Connection Navigator" on page 3-41.

New wizards

Visual Cafe Database Edition contains many new wizards and customizers to help you design, create, build, and deploy your databound project. These wizards support the use of Swing components. The new wizards and customizers include:

- DataBound Project Wizard

  The DataBound Project Wizard helps you create the foundations of your databound project. You specify datasources, tables, columns, and visual components that you'll need for developing your databound project. For more information, see "Creating a databound project" on page 3-5.

- Add Table Wizard

  The Add Table Wizard allows you to quickly add data binding to your project from a datasource. You can use this wizard at any time to add data connectivity and visual components. For more information, see "Adding database tables" on page 3-23.

- Stored Procedure Customizer

  The Stored Procedure Customizer allows you to add database stored procedure support to your projects. Your programs can use stored procedures with parameters and stored procedures that return result sets. Stored procedures that return altered values can be bound to components just like any other data. You can trigger the execution of

stored procedures automatically or programatically. For more information, see "Working with stored procedures" on page 3-67.

- Calculation Customizer

  This feature will help you customize a `Calculation Adapter` to perform common mathematical or string calculations. The resulting data can be bound to visual components dynamically. For more information, see "Calculation and validation rules" on page 3-52.

- Validation Customizer

  The Validation Customizer can be used to validate any input to ensure that it conforms to the specified criteria. For more information, see "Calculation and validation rules" on page 3-52

- Add/Data Bind Component Wizard

  The Add/Data Bind Component Wizard is a quick and convenient way to create and bind data to a Swing component. For more information, see "Using the Add/Data Bind Component Wizard" on page 4-22.

Custom property editors

Many properties for databound components have their own custom editors and customizers. You can use these editors and customizers from the Property List. First, click on a property, then click on the ellipses that appear. For more information, see *Visual Cafe User's Guide*.

Query By Example

Query By Example (QBE) is a convenient and easy way to let the users of your programs retrieve data from a database. The query language used by QBE allows users to create complex queries from a form without having to write complex SQL statements. For more information, see "Creating QBE queries" on page 5-8.

Preface

Welcome to the Database Developer's Edition of Visual Cafe. This book contains all the information you need to take advantage of the powerful databound features of Visual Cafe. This chapter gives you an overview of the documentation for the Standard and Database Editions of Visual Cafe.

Stylistic conventions

This manual uses the following typographic conventions:

- File names, file extensions, directory names, code fragments, and information you type appear in the `code` typeface. Syntactic metanames (items the user must specify, as contrasted with fixed character sequences the user must type) appear in *italic* type.
- The first mention of a glossary item appears in **bold** type.
- All numbers are decimal unless otherwise noted. Hexadecimal numbers are written with the prefix `0x`, as in `0x3EFA`.
- Keys you press at the same time are shown as follows: Control-Z, Shift-Control-G, and so on. You should note that although the letter keys are listed in uppercase, do not hold down the Shift key when executing these key combinations unless the Shift key is listed as part of the combination.

Documentation overview

This section describes the documentation sets for Visual Cafe Database Development Edition. This documentation is in addition to the set that you will need to use the standard features of Visual Cafe.

The ReadMe text file

The ReadMe text file is the first document you should read before using Visual Cafe. It contains late-breaking news, work-arounds, and known issues. This file is available for viewing at the end of the installation process, as well as from the Windows Start menu.

Getting Started book and tour

This book contains all the information you need to start using Visual Cafe Standard Edition and Database Edition. The Visual Cafe tour is an overview of the features of both the Standard and Database Editions. The Tour requires approximately an hour to complete, and requires no programming experience, and it's fun!

User's Guide

This book contains all the information you'll need to use the rest of the standard (non-databound) parts of Visual Cafe. This book includes information on projects, compiling, and debugging your programs. It also provides troubleshooting information for your Visual Cafe installation and an overview of JFC programming.

Sourcebook

The *Visual Cafe ourcebook* is where you'll find cookbook examples that illustrate some complex programming concepts. The source code that is used in this book is available for you to copy and modify.

Database Developer's Guide

This book contains both conceptual and procedural information for developers of data-aware Java programs using Visual Cafe Database Edition.

- Chapter 1, "Welcome" provides an introduction to database concepts and application development in Visual Cafe.
- Chapter 2, "Setting Up Database Access" provides information on creating a database for your databound projects.
- Chapter 3, "Developing a Databound Project" contains information on using Visual Cafe features for creating and working in databound projects.
- Chapter 4, "Data Binding" describes general databinding concepts and using Visual Cafe features to bind components to datasources.
- Chapter 5, "Managing Data and Connections" describes using Query By Example, and masking.
- This manual also contains appendices, a glossary, and an index.

Online Help

From within the Visual Cafe programming environment, you can read the Online Help at anytime by pressing F1. The Online Help is context sensitive and will provide you with information and procedures for any feature of the Standard and Database Editions of Visual Cafe. You can also access Online Help from the Help menu in the Visual Cafe main menu.

Other Online Help systems include:

- Database Reference
- Java API Reference
- Java Language Reference
- Macro Reference

dbANYWHERE Installation Guide

This book contains all the information you need to install, configure, and run dbANYWHERE. dbANYWHERE is a middle-ware server developed by Symantec as a solution for your Java programs to communicate with the most popular databases.

Portable Document Format (PDF)

Portable Document Format (PDF) versions of the books are included with your copy of Visual Cafe. These documents require that Adobe Acrobat Reader be installed. Adobe Acrobat Reader is included on the installation CD-ROM of your Visual Cafe product. It is also freely available from Adobe Systems at www.adobe.com.

Support information

Installation and Symantec Support information is included in the jewel case of the CD-ROM.

How much programming do I need to know?

With Visual Cafe, it's possible to develop Java programs without writing a single line of source code. There are many ways to obtain Java programs, such as books, the Internet, and your friends and colleagues. You can drop these programs into Visual Cafe and have an applet, application, or Bean that's ready to use. However, sometimes these programs need modifications, and that's where real challenges to your programming expertise occur.

This manual assumes that you're familiar with the Java programming language or are learning how to program in Java. It's beyond the scope of this manual to teach Java programming, although you can learn more about Java by using Visual Cafe. To learn how to program in Java, you can consult one of the many excellent books that extensively explore the Java language. You can also search the Internet and find many well-written Java tutorials and summaries, as well as abundant resources to guide you on your way to becoming a Java developer. You might want to participate in a

Java programmers' special interest group (SIG) in your area. You can find user groups on the Web or through Usenet newsgroups.

To use Visual Cafe, it helps to have a basic understanding of object-oriented programming languages, such as C++. Many of the principles and concepts of Java are based on those found in C++, although you should note that there are also some vast differences between Java and C++.

You should also have a basic understanding of cross-platform operating system concepts. This knowledge is useful, for example, when you're developing multithreaded Java programs, because all platforms handle threading differently.

If you're using Visual Cafe Professional Edition, you need to have basic Microsoft Windows programming skills in order to develop native 32-bit applications and libraries. For more information, see Chapter 11 of the *Visual Cafe User's Guide*, which shows you how to create native Win32 applications.

If you're using the Database Edition, a basic understanding of Structured Query Language (SQL) and client-server models is necessary if you want to to build complex databound Java programs.

Finally, a basic understanding of Hypertext Markup Language (HTML) is helpful so you can develop relationships between your Java applets and web pages.

Updating Visual Cafe with LiveUpdate

You can check for updates for your Visual Cafe product by selecting LiveUpdate from the Help menu. For more information about using LiveUpdate, consult the *Visual Cafe User's Guide.*

CHAPTER 1

Welcome

Symantec's Visual Cafe Database Edition is a complete Rapid Application Development (RAD) environment for Java that provides database connectivity and features a sophisticated set of high-level tools.

Information in this chapter includes:

- An overview of Visual Cafe's database tools
- An overview of components in the Visual Cafe Database Edition
- Building a program and deploying it

Overview of Visual Cafe Database Edition

Visual Cafe Database Edition takes Java development to a new level by incorporating two powerful development tools:

- Visual Cafe
- Database Edition Extension

In addition, the Database Edition includes the dbANYWHERE server to support your legacy dbANYWHERE programs and components.

With Visual Cafe, you can assemble complete Java applets and applications from a library of standard and third-party objects without writing a lot of Java code. The Database Edition extensions enable you to bind these Java programs to datasources. Like the Professional Edition, Visual Cafe Database Edition seamlessly integrates visual and source development of Java software, letting you switch between visual and source views. The

two-way editing feature immediately updates the visual environment with any changes you make in the source code. Also, the source code automatically updates changes that you make in the visual view.

Visual Cafe Database Edition offers the same features as Visual Cafe Standard Edition, plus features that allow your application to communicate with a database. Visual Cafe does this by generating code that's compliant with the JDBC API.

The dbANYWHERE server

Alternatively, you can use the dbANYWHERE API, which was used in Visual Cafe Database Development Edition before version 2.1 and in Visual Cafe Professional, and is supported by the dbANYWHERE server.

The dbANYWHERE server is a tool that manages connections between a Java program and one or more databases. The dbANYWHERE server provides a 100% pure JDBC solution as well as a proprietary dbANYWHERE API.

It is included with this version of Visual Cafe to support any components that were created before JDBC-Beans were available.

Primary features of the Database Edition

The main features of the Database Edition are:

- JDBC support
- Databound wizards
- The dbNAVIGATOR window
- Databound components, including support for databound Swing components

JDBC support

Visual Cafe generates JDBC code to connect your project to a database. The JDBC standard set of classes and interfaces were developed by Sun to extend the Java language to data connecvity.

The Database Edition has added some of its own classes and interfaces to the JDBC standard set of classes and interfaces. These extensions allow your application or applet to “talk” to a database and “share” information with it.

JDBC adds connectivity to Java API by executing SQL statements. By using it through your Visual Cafe program, you can communicate with virtually any relational database. In short, it carries forward the Java promise of “write once, run anywhere” into the realm of relational databases.

What can JDBC do?

By creating an applet or application using Visual Cafe to generate JDBC code you gain the ability to:

- connect to a database.
- send SQL statements to query your database.
- generate query results.

Examples of databound applets might be a corporate phone list or an e-mail form which allows you to receive feedback from a Web page. A databound program might be a stock reporter, which gathers information from a database on the Web and stores it in a database on your local machine.

Databound wizards

The Visual Cafe Database Edition wizards help you create the framework for database projects. The tasks they simplify include:

- creating a new form based on a datasource table or stored procedure
- adding datasource table connectivity to an existing form
- defining master-detail joins

- adding support for stored procedures and business rules

The Database Edition wizards create code based on JDBC Beans. These Beans work with JDBC-compliant datasources to accomplish the tasks of your program. Once you use the wizards to define the relationship between your datasource and components in your applet or application, Visual Cafe automatically updates AWT DataBound components in your form to match the changes.

The DataBound Project Wizard

You can use the DataBound Project Wizard to create a project that connects to a datasource. It provides dialog boxes where you're asked to make choices that define your project. The wizard helps you define the datasource, choose the table, specify columns, and add new components, such as navigation buttons, status bars, panels, layouts, and interactions. When you finish, the wizard displays the form you specified in the Visual Cafe Form Designer. For more information, see "Creating a databound project" on page 3-5.

The Add Database Table Wizard

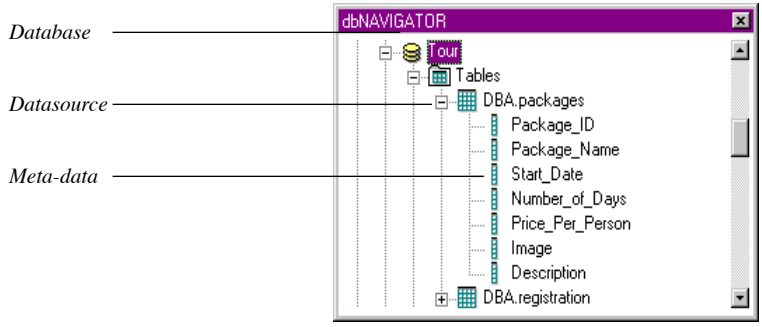
After you create your form, you can add more tables to it by using the Add Database Table Wizard. The Add Database Table Wizard allows you to:

- Create a new form based on a datasource table or stored procedure
- Add datasource table connectivity to an existing form
- Define master-detail joins
- Map columns to form components

Once you use the wizards to define the relationship between your datasource and components in your applet or application, Visual Cafe automatically updates AWT DataBound components in your form to match any changes in the datasource. For more information about AWT DataBound components, see "AWT DataBound components" on page 1-6. For step-by-step instructions on using the Add Database Table Wizard, see "Adding database tables" on page 3-23.

The dbNAVIGATOR window

The dbNAVIGATOR window provides a hierarchical view of available datasources. The dbNAVIGATOR window shows the databases that you're connected to, tables within the databases, and the columns within the tables. This database information is commonly referred to as *metadata*.



You use dbNAVIGATOR as a way of cataloging meta-data. dbNAVIGATOR lets you see all connected database datasources. Datasources may be stored procedures, dbANYWHERE servers, and the tables and columns in a database. You can also view JDBC-ODBC bridges.

In the following illustration, you can see the metadata for the Tutorial database. This database is used with the Tour in the *Getting Started* book.

After connecting to your database, you can use dbNAVIGATOR to:

- navigate across the servers that you're connected to and the databases managed by the servers.
- expand and contract the datasource listing and use drag-and-drop techniques to build your databound forms.
- view database contents in a hierarchical manner.
- drag database tables and/or fields onto your application forms. When you drop a database object onto your form, Visual Cafe creates the associated components and populates the component Property List with information collected from the database.

Databound components

Visual Cafe Database Edition includes a variety of databound components. This section provides an overview of the components that you can use to develop your databound projects:

- AWT DataBound components
- Data Source components
- Swing DataBound components

These components simplify database connectivity and facilitate binding to database columns. Once data is bound to forms, they can communicate with **datasources** through a JDBC server. Each of the AWT DataBound, Data Source, and Swing DataBound sets of components have similar data-binding processes. For more information, see "Data binding in Visual Cafe" on page 4-12.

AWT DataBound components

An AWT DataBound component is a Visual Cafe component that has additional properties for binding it to data in a database. AWT DataBound components provide the built-in functionality you need to display, communicate with, and connect to a database.

These components are designed to save you extensive programming time and effort. You can quickly add AWT DataBound components by using the DataBound Project Wizard (File, then New Project) and the Add Database Table Wizard (Database, then Add Database Table Wizard, or double-click the Add Database Table Wizard icon on the Component Palette.)

AWT DataBound components are bound to the current row in a single database column and some can be bound to another database column for component population. Some AWT DataBound components provide projection binding, others provide list binding. For more information on data binding, see "Data binding in Visual Cafe" on page 4-12. Several of the standard Visual Cafe AWT components have an equivalent AWT DataBound component.

For example, you can set an AWT DataBound `TextField` component to correspond to a table column. When you do this, the text reflects the value stored in that database table column.

You can read detailed information about these components in the Online Help, which provides information about the properties of each component. You see the DataBound components by clicking on the AWT DataBound tab of the Visual Cafe Component Palette.

Some of the AWT DataBound components in the Visual Cafe Database Edition are:

- `RadioButton`

  Used to select one choice from several options. When you add the first AWT DataBound `RadioButton` to a project, a `RadioBox` component is automatically created. The `RadioBox` component is used to bind `RadioButtons` to the database. The column value is used as the `RadioButton` component label value.

- `Mediator`

  A JavaBean that can be used with any AWT component to make it become a databound component. A `Mediator` is a bridge between the component and the database. It can transmit single cell values as well as multi-dimensional arrays of cells.

- `RecordNumberLabel`

  At run time, this component displays the current record's number in the retrieved record set.

- `DBCurrencyTextField`

  A text field that automatically applies currency formatting logic to user input.

- `ListPlus`

  A scrollable and selectable listbox of items populated from a datasource.

- `Label`

  AWT DataBound labels function like a `TextField`, but are not editable. The text string for the label is obtained from the database column at run time, according to criteria in your code.

- `TextArea`

  An editable multi-line display window with horizontal and vertical scroll bars. The display window can scroll through an amount of text that is too large to display at one time. Use this component to display long strings or text data from a database.

- `RecordStateLabel`

  At runtime, this component displays the state of the current record.

- `FormattedTextField`
  This component allows you to control the display of a column value by using a text mask to specify the format.

- `TextField`
  A box in which you can type text. Use this component to collect text or to display strings from your database. The `TextField` component can be resized, but doesn't have scrollbars.

- `Grid`
  Displays multiple data records. For information about using the Grid component, "Adding a Grid component to your project" on page 3-44.

- `TotalNumberofRecordsLabel`
  At run time, `TotalNumberofRecordsLabel` automatically indicates total number of data rows in the database table or result set.

- `DBTstamp`
  A box in which a date or timestamp can be displayed or entered by a user. It supports all common date and time formats.

- `RadioBox`
  A container which groups `RadioButton` components. The `RadioBox` component allows you to bind `RadioButton` components to a database column.

- `ImageViewer`
  Displays images that are retrieved from a database column or loads an image from a non-database source, displays it, and then stores the image in the database. Currently only the JPG image format is supported.

- `DBStatusBar`
  Provides information about a database task that an application is running. The status bar also provides a custom handler for the errors raised by the application.

- `DBDateMaskedField`
  Applies mask formatting logic to the user input.

- `CheckBox`
  Displays a check when selected and nothing when deselected, One use for an AWT DataBound `CheckBox` is to link a `CheckBox` and `ListBox` to the same column, and define `true` and `false` string values for the `CheckBox`. Then, when a user selects the `CheckBox` option, the

`true` value displays in the list box; when the `CheckBox` is cleared, the `false` value displays.

- `ComboBox`
  Creates a text box with an attached list box. A `ComboBox` can be populated with data from a datasource.

- `DBToolbar`
  Displays a toolbar that controls database table navigation.

- `NervousText`
  Creates animated text in which each letter moves independently of all other letters. The text for this component is retrieved from a datasource. This multimedia component is provided for novelty and programming convenience.

- `dbMaskedTextField`
  Applies mask formatting logic to the user input.

Data Source components

Some of these non-visual components are created automatically in your Visual Cafe project when you define datasources. Other components, such as the adapters, are added and customized when you use the respective wizard. You can also drag and drop Data Source components and then run the respective wizard to customize them.

- `SQL Adapter`
  Creates a framework into which you bring SQL statements for your project. After you define the properties for a `SQL Adapter` in the Property List, the visual editors allow you to implement SQL statements in your project.

- `Query Navigator`
  Manages a set of records using the JDBC API. Dragging a Data Table item from the dbNAVIGATOR to a form in a project creates a `Query Navigator` object and a top-level `Record Definition` object. The `Query Navigator` supports Query By Example (QBE), including sorting and the SQL `WHERE` clause.

- `Stored Procedure Adapter`
  Presents a collection of SQL statements that are named and precompiled. The `Stored Procedure Adapter` allows you to add a stored procedure from a database to your project.

- `Stored Procedure Adapter Trigger`

Listens to the `Query Navigator` for changes in data, then tells the `Stored Procedure Adapter` to execute the stored procedure in the database.

- `Calculation Adapter`

  Creates a calculation that you specify and displays it in a text-based component. Some of the available functions include addition, subtraction, multiplication, and summations.

- `Validation Adapter`

  Used to compare data that a user inputs with the data definitions from a datasource.

- `Jdbc Connection`

  Creates a JavaBean that represents a single JDBC connection. It has properties for establishing connection parameters in accordance with JDBC. Dragging a Data Source item from dbNAVIGATOR to a project creates a `Jdbc Connection` object. You have one `Jdbc Connection` object per datasource. All `Jdbc Connections` in a project are managed by a single `Connection Manager` object.

- `SQL Record Definition`

  Represents a row in the result set of user-defined SQL. `SQL Record Definition` is a non-visual component at project level.

- `Connection Manager`

  The `Connection Manager` manages all the connections (`Jdbc Connection`) of an application. It is a top-level non-visual Bean, meaning that it is not contained within any form.

- `Record Definition`

  Defines and accesses a row of data in a database table. It is a top-level non-visual Bean.

For more information on any of these components' properties, see the Online Help.

Swing DataBound components

The Swing components included with Visual Cafe are JFC components that are capable of providing powerful functionality to your Java program. The Swing DataBound components enable standard Swing components to bind with a datasource. Some or all of these components will be deployed with your project.

- `TableBindingModel`
  Defines data binding for a `JTable`. For instance, if you want to create a table that displays data from a datasource, you must first drop a `TableBindingModel` into your project.

- `BinderModel`
  Defines data binding when you use JFC components other than `JTable`. The DataBound Project- and the Add Database Table Wizards automatically attach this model when using the following JFC components: `JTable`, `JList`, `JCheckBox`, `JTextField`, `JTextArea`, `JComboBox`, `JRadioButton`, or `JLabel`.

- `JDBToolbar`
  Contains buttons to perform database functions, such as `Next` and `Previous`.

- `JRecordNumberLabel`
  Indicates the current data row in the database table or result set.

- `JDBStatusBar`
  Contains components which display data row status, such as record number and record state. `JDBStatusBar` also displays the information about a current process, and contains custom handlers for errors that are raised by the program.

- `JTotalNumberOfRecordsLabel`
  Indicates total number of data rows in the database table or result set.

Data Connection Navigator

The Data Connection Navigator (dcNAVIGATOR) is a tree control that lists all the current datasources. The dcNAVIGATOR is available from several places in Visual Cafe. It appears as an alternative view of the datasources in a project. In addition, it can be launched from from a Browse button or an ellipsis (...) button whenever you need to select a datasource object such as a `Query Navigator`. The dcNAVIGATOR is also accessible from the Property List when you click a datasource.

dcNAVIGATOR helps you stay informed, provide consistency with the dbNAVIGATOR tree and facilitate the selection of hidden data connection objects.

About data binding

When you want to display data from a database to a user, you must first bind the datasource to a visual component. There are several ways to accomplish data binding depending on what kinds of components and datasources you are using.

When you use the DataBound Project Wizard to create your project, you can visually bind data components with datasources. You can also use drag-and-drop techniques to bind components and datasources.

Database manipulation functions

Visual Cafe provides a set of functions you can use to navigate and manipulate your database. Some of these are provided in buttons that you can add to forms. There are also several database methods that don't have a visible representation.

Visual Cafe includes the following database functionality:

- Next record: displays the contents of the next database record
- Previous record: displays the contents of the previous database record
- New record: creates a new record in the database
- First record: displays the first record
- Delete record: deletes the current record
- Undo record: undoes changes made to a database record
- Save entire record set (save multiview): saves a set of records
- Rewind result set: returns to the beginning of the result sets
- Restart entire record set (restart multiview): starts over at the beginning of a set of records
- Goto record: goes to the record specified by an integer
- Get record state: reports the state (modified, unchanged, deleted, new) of the current record
- Query By Example (QBE): creates query from your form using data in a field as an example.

Building a databound project

When you're ready to develop your databound project, there are several steps you must follow before the application or applet can read and store data.

To process a databound project:

1. Create a databound project using the DataBound Project Wizard.
2. Use the Add Database Table Wizard to add new tables from your database (optional).
3. Add the databound components your require. These components may include calculation and validation rules, stored procedures, and images, for example.
4. Compile, test, debug, and deploy your project.

   For information on creating a data-bound project, see "Creating a databound project" on page 3-5.

Deploying your database project

Deploying applets and applications developed with Visual Cafe Database Edition is the same as deploying Visual Cafe applets and applications. See the *Visual Cafe User's Guide* for more details.

Note that there are additional classes for the Database Edition. The directory `bin\components` contains `databind.jar` and `symbeans.jar`.

C H A P T E R 2

Setting Up Database Access

The Visual Cafe Database Edition contains many features that enable you you to quickly and easily develop and deploy databound Java programs.

This chapter describes:

- Tools you need and how they interact to provide database access
- How to configure these tools and test your configuration
- How to test your database connectivity

Installing the software for database development

To develop databound Java programs with Visual Cafe, you must first ensure that all the necessary tools are installed. The tools you need are:

- Visual Cafe
- JDBC driver
- Database engine and client

These tools can be installed on the same computer or on different computers that are connected through a TCP/IP network.

Installing Visual Cafe

The procedure for installing Visual Cafe is described in the Visual Cafe CD liner. When you install Visual Cafe, make sure you install all components.

Installing the JDBC driver

Before Java emerged, Open DataBase Connectivity (ODBC) was the standard interface for communicating with databases. A Java Database Connectivity (JDBC) driver provides a communication channel between Visual Cafe and the database. Visual Cafe does not communicate directly with the database; it uses a JDBC driver to do so.

You can install the JDBC driver locally (on the same computer as Visual Cafe) or on a remote computer that can communicate with the local computer through TCP/IP.

The Visual Cafe CD contains a tool called the dbANYWHERE server that provides the services of a JDBC driver. The dbANYWHERE server also provides connection management, data buffering, and a robust API that extends JDBC with enhanced data access features. For more information, see the *Installing and Running dbANYWHERE Server* manual.

Alternatively, you can use JDBC-ODBC Bridge or some other proprietary solution. The JDBC-ODBC Bridge is automatically installed as part of the Java Development Kit (JDK). For more information on using alternative JDBC drivers, see "Configuring the JDBC driver" on page 2-3.

Installing the database and client software

You can install a database engine locally (on the same computer as the JDBC driver) or on a remote computer that can communicate with the local computer through TCP/IP. Remote database engines (and some local engines) also require client software that must be installed on the same computer as the JDBC driver.

The procedures for installing database and client software vary from vendor to vendor. For detailed instructions, refer to the documentation that accompanies your database product.

Configuring data access

After installing all the necessary tools, you need to configure them so they can communicate with each other.

Launching the database development tools

The first step is to launch the tools you need for database development. You can launch Visual Cafe the same way you launch most Windows programs: from the Windows Start menu.

If you're using dbANYWHERE Server as your JDBC driver, you also need to launch it. If you've installed dbANYWHERE on a machine that is running Windows NT, you can have it start automatically when its services are requested. For more information, refer to the *Installing and Running dbANYWHERE Server* manual.

Starting the database engine

The procedures for starting the database engine vary from vendor to vendor. For detailed instructions, refer to the documentation that accompanies your database product.

Configuring the JDBC driver

Visual Cafe and Java programs use a JDBC driver to communicate with the database. Instructions for using dbANYWHERE or the JDBC-ODBC Bridge for a JDBC driver are provided below.

Alternatively, you can use another third-party JDBC driver. For more information on setting up other JDBC drivers, refer to the documentation that accompanies the product.

Using the JDBC-ODBC Bridge

The JDBC-ODBC Bridge is part of the JDK, which is installed (by default) when you install Visual Cafe. The JDBC-ODBC Bridge converts the JDBC methods in your Java programs to ODBC functions. It allows you to use most ODBC data sources registered on your computer. These datasources include Oracle, Informix, Sybase, to name a few.

To create an ODBC data source:

1. Launch the ODBC Administrator utility from the Windows Control Panel.
2. Click Add and follow the directions for creating the data source.

For more information on creating data sources, click the Help button.

Using the JDBC-ODBC Bridge does not require any extra configuration, but it is has some limitations:

- It does not work with downloaded, unsigned Java applets, because of security violations.
- Java applications that use the JDBC-ODBC Bridge require the presence of the client software on the local machine.

You can find more information about the JDBC-ODBC Bridge on the JavaSoft Website, `http://java.sun.com/products/jdbc/`.

Using dbANYWHERE

To use dbANYWHERE as your JDBC driver, you need to create a data source for each database you want to access through dbANYWHERE. You can do this using the Data Source Tool provided by dbANYWHERE. The *Installing and Running dbANYWHERE Server* manual explains how to create and test data sources for dbANYWHERE.

Once you've created the data source, you're ready to use that database in Visual Cafe. See the *Installing and Running dbANYWHERE Server* manual for more information on configuring database access through dbANYWHERE.

Testing the connection

After Visual Cafe, the JDBC driver, and the database are up and running, you can create a database and test the accessibility of the data.

- If you have an existing database to work with, you can create a datasource and use it to test the connection.
- If you installed the Getting Started tour when you installed Visual Cafe, you can use the tour's database to test the connection.

You can find the instructions for creating the Getting Started tour's database and data source in the file `createdb.txt` in the `Tour` folder of your Visual Cafe installation folder.

- If you have neither of the above, you can use the tools that accompany your database product to create and populate a test database. For information on how to do this, refer to the documentation that accompanies your database product.

Using dbNAVIGATOR

The following procedure allows you to quickly test your configuration using dbNAVIGATOR.

To verify your configuration:

1. Launch your database engine.

   If you have trouble launching the database engine, refer to the documentation that accompanies the database product.
2. If you're using dbANYWHERE as your JDBC driver, launch it.
3. Launch Visual Cafe.
4. Choose dbNAVIGATOR from the View menu.

   A dbNAVIGATOR window appears.
5. When the server appears in the Data Sources window, click the + next to the server icon to display the databases registered to that data source.
6. Click the close box.

If you can see the database under the connection test server, your data is accessible. If, however, you were unable to complete this procedure, see "Troubleshooting your connection" on page 3-11.

CHAPTER 3

Developing a Databound Project

When you create databound Java programs with Visual Cafe, you develop them in **projects**. A project can contain multiple applets and applications, as well as HTML files. For more detailed information about working with projects, see Chapter 2 of the *Visual Cafe User's Guide*.

Visual Cafe Database edition offers wizards, editors, and customizers to help you create a databound project. When you use these features, you quickly identify datasources for your program and add databound user interface elements such as buttons, text fields, and combo boxes. Then, you specify events and interactions that bind your visual components with the data.

An example of a databound project is e-mail that's generated from your Web site. This e-mail might contain a customer order to be stored in your database. Another project might be a newsreader or stock ticker application that stores its data in a database that could be located anywhere in the world.

This chapter describes the basic procedures involved in developing a databound project using Visual Cafe. The procedures described here in this chapter assume that you have access to a database through a JDBC driver, as explained in "Configuring the JDBC driver" on page 2-3.

Before you create your databound project

A databound project contains an applet or application that accesses a database. When you are ready to develop your databound project, there

are several steps you must follow before your Java program can read and store data.

1. Either use an existing database or create one by using the tools that are included with Visual Cafe. For more information, see "Launching the database development tools" on page 2-3.
2. Identify the database you want to use for the project's datasource by using the ODBC Administrator Tool. This tool is installed along with Visual Cafe and appears on the Windows Start menu. For more information, see "Launching the database development tools" on page 2-3.
3. Create a databound project using the DataBound Project Wizard. For more information, see "Starting the DataBound Project Wizard" on page 3-5.
4. Use the Add Table Wizard to add new tables from your database. For more information, see "Adding database tables" on page 3-23.

Creating the database for your project

Before you can develop a databound project you must have a database in place for your project's data. If you need to create a new database for you project, you can use:

- The Oracle Lite set of database creation tools, which are included with your Visual Cafe Database Edition package.
- Proprietary database tools for the SQL Sybase, Informix, and Oracle databases

Defining a datasource

A **datasource** is an object that identifies a database to a JDBC- or ODBC-compliant database application. (**ODBC** stands for Open DataBase Connectivity). It is the standard interface to database applications in the Microsoft Windows environment.

When you create a datasource for your project, you're packaging the information that enables your application to connect to the database you want to access. The datasource includes details that are needed for connection, such as the name of the database, its server, and its network location.

Before you can use your database in a Visual Cafe project, you must specify its name. You create a datasource name using the ODBC Administrator which is available in the Windows Control Panel folder.

If your database uses one of the "flat file" database systems such as MS Access, SQL Anywhere, Watcom, or ODBC XBase as its driver, you must use the ODBC Administrator program to create its datasource.

To add a datasource using the ODBC Administrator:

1. From the Start Menu, choose Settings, then Control Panel, then ODBC.

   The ODBC Administrator appears.

2. Complete all pages in the wizard to add a new datasource.

Setting database environment options

When you install the Visual Cafe Database Edition, the Database tab is added to the Environment Options dialog box. You use this tab to access a list of the components that will be associated with particular data types.

You can also specify certain environment options that determine how code is generated for your databound projects. You can:

- specify Swing component assignments for dataypes
- specify non-Swing (AWT) component assignments for datatypes

These settings apply to code that is generated after the option is set.

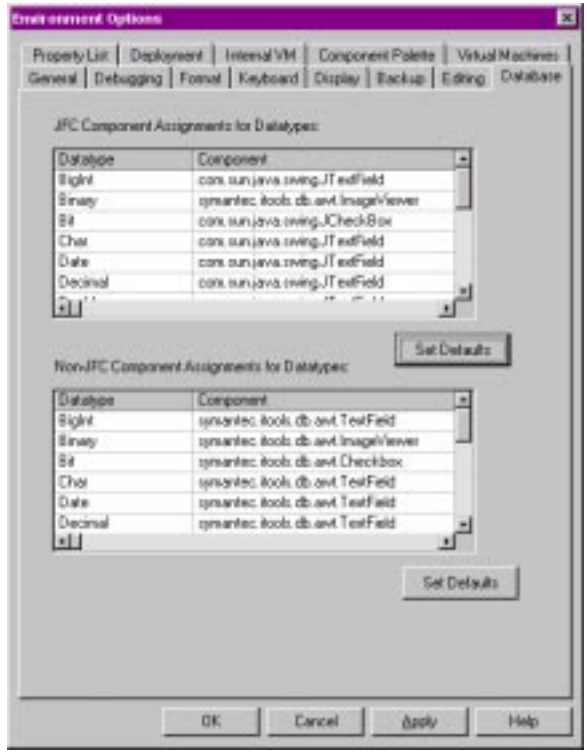
Note: The Environment Options dialog box is available only when a project is open.

To specify the component to generate for a datatype:

1 Choose Environment Options from the Tools menu.

The Environment Options dialog box appears.

2 Click the Database tab to display the database environment options.



3 Click a datatype's Component field to see a drop-down list of available components.

4 Click Apply to apply the settings. If you want to apply the changes and close the dialog box, click OK.

You can reset the list to the Visual Cafe defaults at any time by clicking Set Defaults.

Creating a databound project

One way you create a databound project is with the DataBound Project Wizard. It includes a series of interactions to help you identify database tables, variable in the tables, and the actions that are to be performed on that data, and how you want to display the results. A button corresponds to each action, and a field and label to each column in the table. Your Java program is constructed and displayed in the Form Designer.

The following section describes using the DataBound Project Wizard, including:

- Starting the DataBound Project Wizard
- Defining datasources
- Selecting tables or stored procedures
- Choosing database columns
- Selecting visual components
- Selecting additional components
- Reviewing your selections

You can click Back at any time while working through the ward to review and make changes to previous pages.

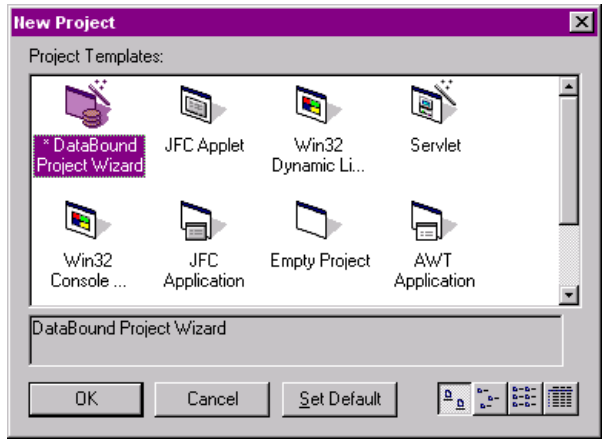
Starting the DataBound Project Wizard

You start the DataBound Project Wizard just like you would for any other Visual Cafe project. The DataBound Project Wizard includes a series of interactions that allows you to select datasources, tables, columns, components, and layouts for your project. At the end of the wizard, the applet or application is constructed and displayed in the Form Designer.

To start the DataBound Project Wizard:

1 From the File menu, select New Project.

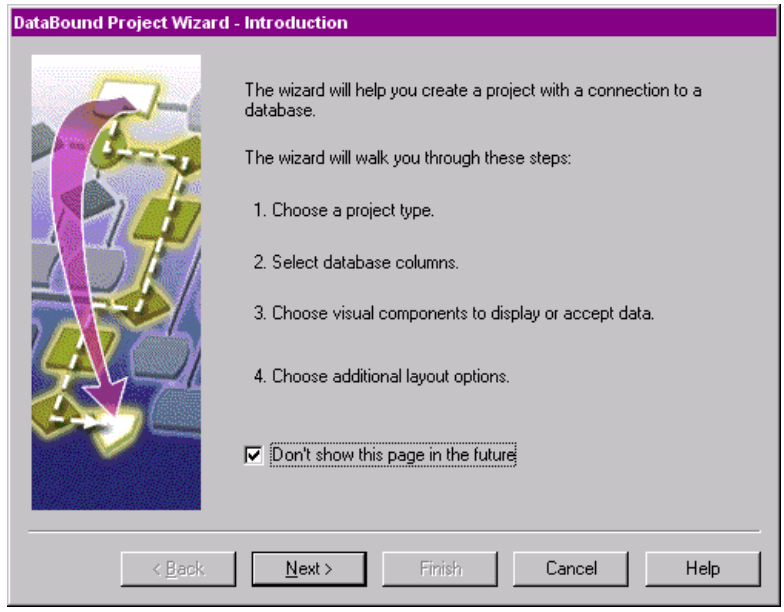
The New Project dialog box appears:



2 Double-click DataBound Project Wizard to launch the wizard.

The Introduction page of the DataBound Project Wizard appears.

The Introduction page provides an overview of the steps involved. You can turn this page off by clicking the checkbox at the bottom of the page.

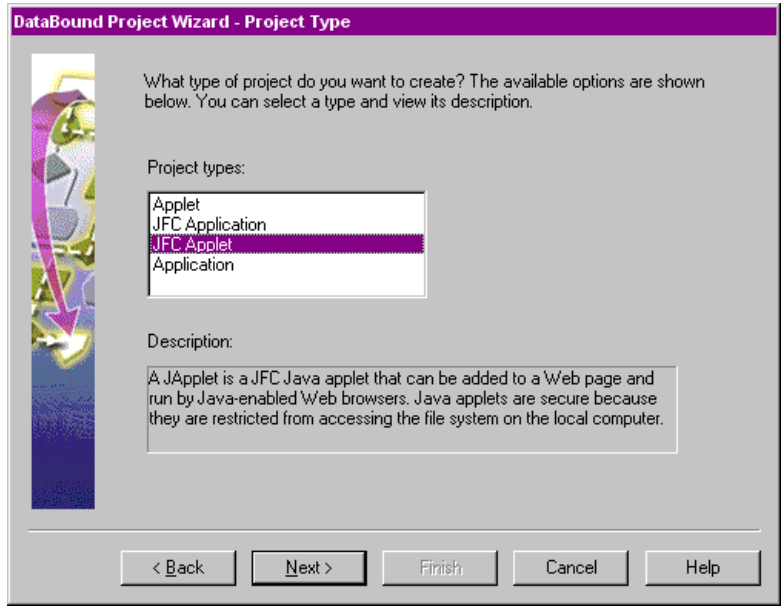


Setting up your project as an applet or application

The Choose a Project Type page is where you select the kind of database project you want to create. When you highlight one of the project types, a brief description appears at the bottom of the page.

To create your project as an applet or application:

1 Choose Applet, Application, JFC Applet, or JFC Application from the Project Type's drop-down menu.

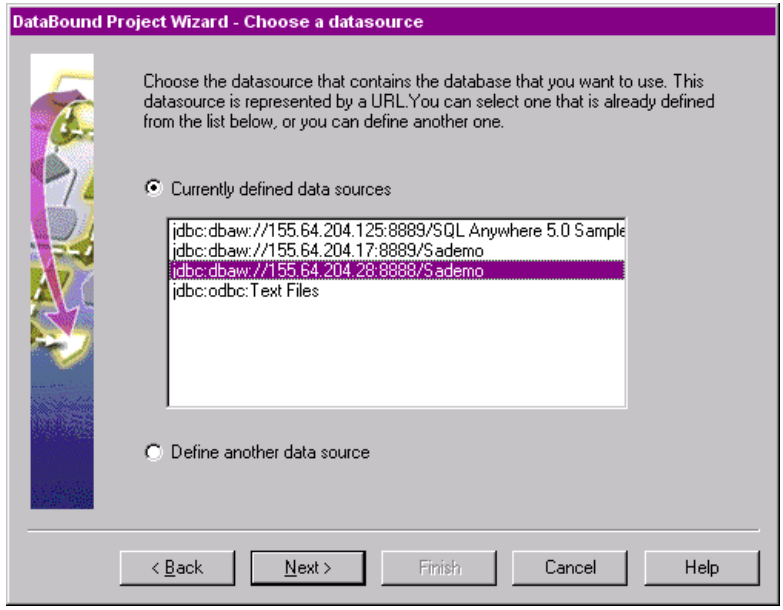


Option	Description
Applet	A program that can be added to a Web page and run by Java-enabled Web browsers (or the AppletViewer).
JFC Applet	An applet built using Swing components. JFC applets use `Japplet`.
Application	An extension of `Frame`, the class created from the Application template, is a good base class for a main application window. It can exist by itself, contain components and menus, and be used to show dialogs and windows.
JFC Application	An application built using Swing components. A JFC application extends `JFrame`.

2 Click Next to select a datasource for your project.

Identifying the datasource

The Choose a datasource page creates the datasource connection for your project.



To select a datasource that is already connected:

- Click a name from the Currently defined datasources.
- Click Next to select a database table or a stored procedure.

To choose a new datasource:

1. Click Define another datasource.
2. Click Next.

   The Insert a Datasource page appears.

   For more information, see "Inserting a datasource" on page 3-11.

Troubleshooting your connection

If you're having trouble connecting to a datasource, consider the following questions:

- Do you have the correct IP address and port number?
- Do you have permission or rights to use the database?
- Is the host machine available? (try a `ping` utility.)

Inserting a datasource

In the Insert a Datasource page, you can select an existing database from the list of currently defined database drivers. If you want to add one that is not in the list, click New. When you enter the name of a valid datasource, that datasource name is automatically added to the Database Drivers list so that you can select it in other projects.

When you click New from the Define a Datasource page, the Insert Datasource page appears. You can also view and specify information about the datasource drivers, such as an IP address, port number, and a list of known datasources. When you add or change information, click Refresh to see the additions or changes you've made.

To insert a datasource:

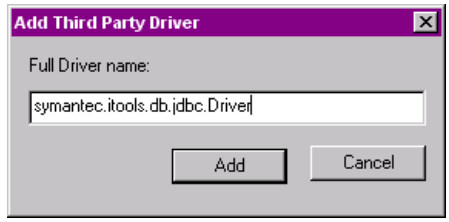
1. Click the appropriate database driver from the Database Drivers list.

   The information about the database appears in the fields, with a list of available data sources.
2. Select a datasource.
3. Click OK to close the Insert Datasource dialog box and add it to the Currently defined datasources list.

To add a third party driver:

1 Click Add Driver.

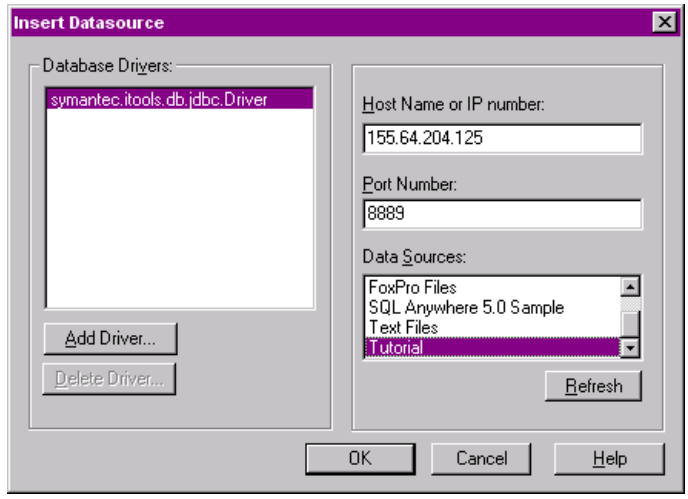
The Add Third Party Driver dialog box displays:



2 Enter the driver's fully qualified name in the dialog box.

3 Click Add.

The database driver appears in the Database Driver window:



4 Select the newly added driver.

5 Enter the host name or IP address for the server.

6 Enter the port number for the server.

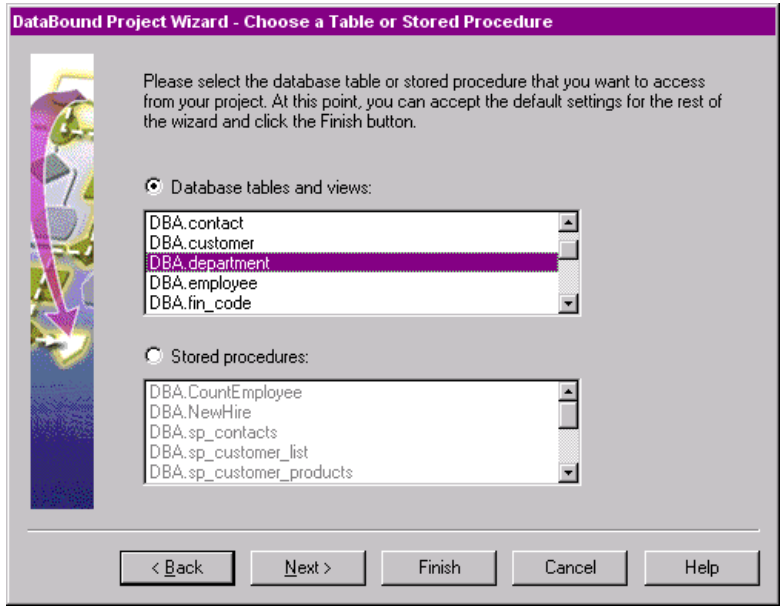
7 Enter the user-defined server name.

8 Click OK to add the server to the Currently Defined Datasources list.

You'll be prompted to log on to the datasource if required.

Selecting tables or stored procedures

Databases can be thought of as a collection of tables. The tables in the database you selected earlier are displayed in the Choose a Table or Stored Procedure wizard page, as shown in the following illustration.



To make a table or a stored procedure and its data available to your Java program, select the table(s) or stored procedures from the appropriate list and click Next.

To select a table:

1. Click a table from Database tables and views. The list displays the database tables that are connected to the datasource you selected previously.
2. Click Next to advance to the Choose Columns page.

To select a stored procedure:

1. Click a stored procedure from the Stored Procedures window.
2. Click Next to advance to the Choose Columns page.

Note: If you have not defined a datasource, the Stored Procedures list will be empty.

Choosing database columns

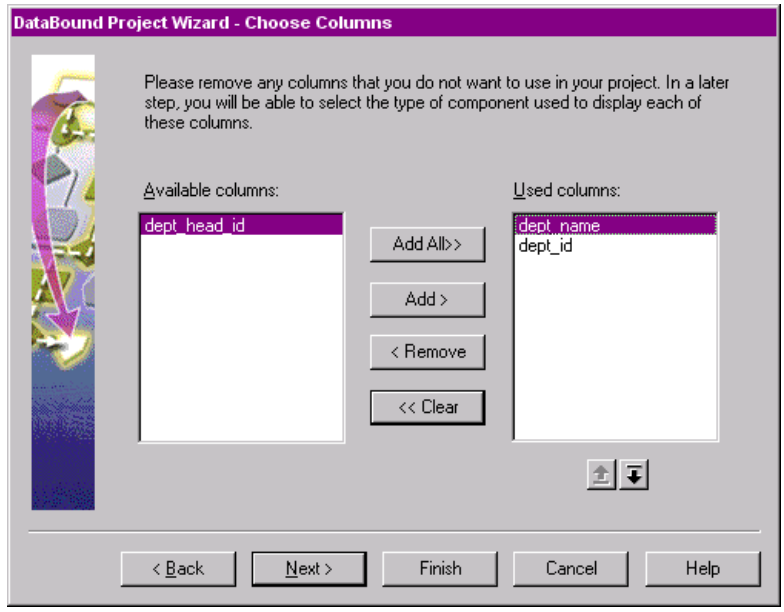
Typically, each column of a database contains a single kind of information. For example, one column of a customer database might contain customer identification numbers, while another column contains a customer name, and yet another contains purchase information.

The Choose Columns wizard page lets you select the database columns that you want to use in your Java program. By default, all the available columns from the defined datasource appear in the Used Columns window. When you deselect columns that you don't want to use, they appear in the Available Columns window.

To deselect database columns:

1. Click the column from the Used Columns that you don't want to use.
2. Click Remove to move the selected columns to the Available columns window. Clear moves all the used columns.

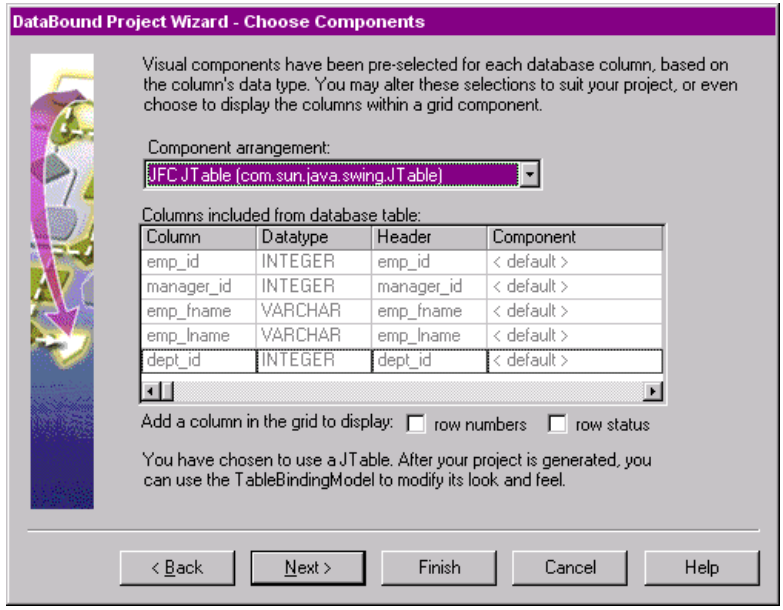
3 Reorder the columns if necessary by clicking Up or Down.



4 Click Next to select visual components for your form.

Selecting visual components

The Choose Visual Components page lets you select a component and specify a label for each database column, or specify that the columns appear in a single grid, which provides the users of your program with a two-dimensional view of data.



To choose a grid:

1. Select JFCJTable or Symantec Table from the Component Selection drop-down list.
2. Check the boxes for Row Numbers and Row Status if you want these column attributes to appear in your grid.

Changing individual component properties

At run time, your applet or application displays each column's data in its designated component and displays a label for the component. The Choose Visual Components page displays a default component and label for each column.

To change individual component properties:

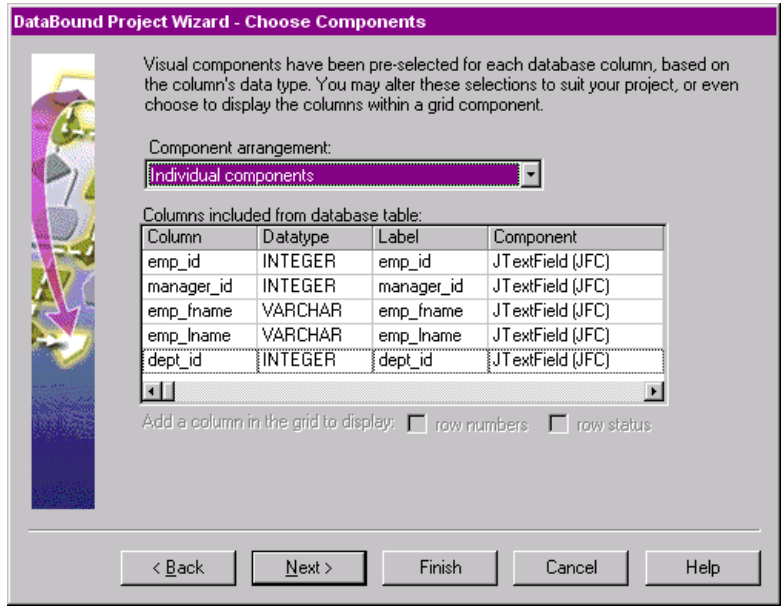
- Select the appropriate individual components from the Component Selection drop-down list.

To change a column's component:

- Click Component, then choose a component from the list that appears.

To change a column's label:

1 Click Label, then type the new text.



You can also change the component's label after it is built by modifying the component's `Text` property. If you don't want the component to have a label, click the text for the label and delete it.

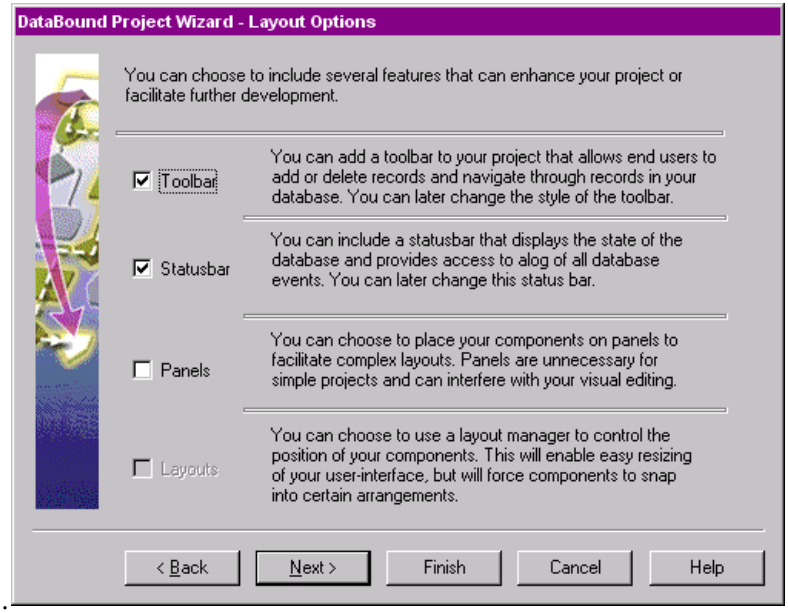
Note: if you choose to use a Grid or Symantec Table, the Label column changes to Header, and you won't be able to modify it in the DataBound Project Wizard.

---

2 Click Next to select additional layouts.

Selecting additional layouts

The Layout Options page lets you select additional components and layouts that can enhance your Java program.

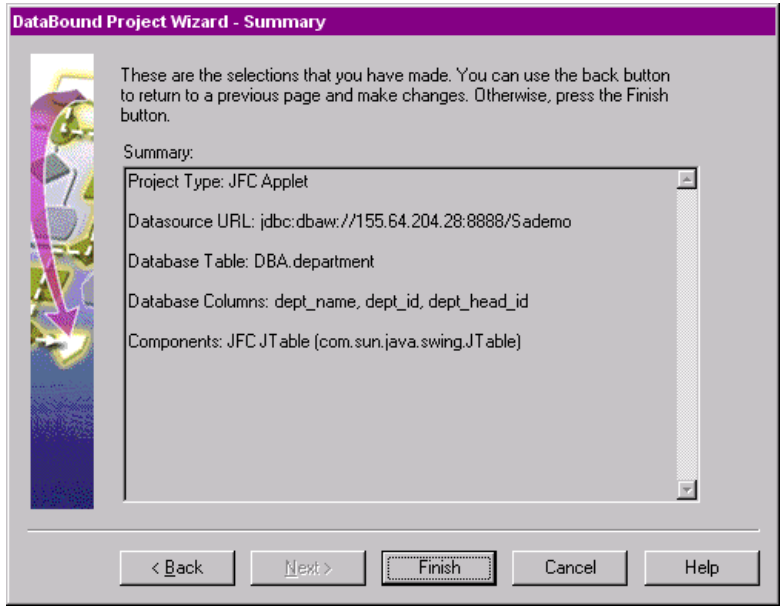


To select additional components and layouts:

- Click the checkbox for each option you want to use on the form. The options include Toolbar, StatusBar, Panels, and Layouts.

Reviewing your selections and creating the project

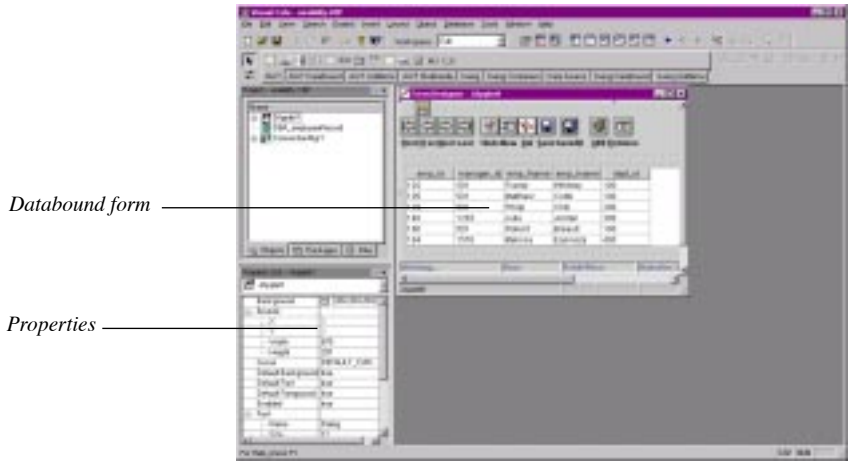
Finally, you can review your selections in the Summary page before creating the skeleton of your databound project. If you'd like to change anything, click Back to make modifications to the appropriate page.



To have Visual Cafe create your project:

1. Review the information about the selections you have made.
2. If you need to change anything, click Back to return to the appropriate page and change your information.
3. If you're satisfied with the selections you've made, click Finish.

Your new databound project might look something like this:

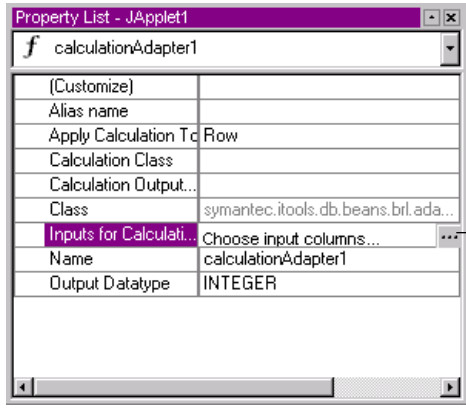


When you finish, the information on the Summary page is reflected in the Property List for this project. If you need to change any of these properties, you can use the Property List to do so. For more information on changing the properties of a component, see "Modifying properties."

The project that is pictured above was made by accepting the defaults on each wizard page. The toolbar that appears at the top of the form can be customized by right-clicking on the `threeDButtonsModel` icon to open the customizer for it.

Modifying properties

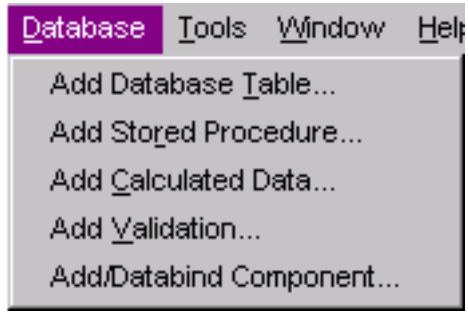
When you need to modify a the properties of a `Binder Model`, `Stored Procedure`, `Calculation`, or `Validation Adapter`, you can use use the Property List. Many properties have their own visual editors. Properties that are modifiable with their own editors are marked with ellipsis. To open the editor for that property, click the ellipsis.



Click the ellipsis to open the available property editor

About the Database menu

When you install Visual Cafe Database Edition, a Database menu appears in the Visual Cafe main toolbar:



Menu Item	Result
Add Database Table	Opens the Add Table Wizard. The Add Table Wizard lets you add more database tables to your project. For more information, see "Adding database tables" on page 3-23.
Add Stored Procedure	Opens the Add Stored Procedure Wizard. You can add support for database stored procedures with the Add Stored Procedure Wizard. For more information, see "Working with stored procedures" on page 3-67.
Add Calculated Data	Opens the Add Calculated Data Wizard. You can incorporate client-side calculation rules into your project with the Add Calculated Data Wizard. For more information, see "Calculation and validation rules" on page 3-52.
Add Validation	Opens the Add Validation Wizard. You can incorporate client-side data validation with the Add Validation Wizard. For more information, see "Calculation and validation rules" on page 3-52.
Add/Databind Component	Opens the Add/Databind Component Wizard. After adding a Swing component to your project, you need to bind it with a datasource, and the Add/Databind Component Wizard helps you with this task. For more information, see "Using the Add/ Data Bind Component Wizard" on page 4-22.

Adding database tables

You will probably need to assign more than one table to a project. For example, a bookstore might keep its pricelook up in one table, ISBN information in another, and inventory in yet another. The advantage of this table organization is that users of such a system can access and sort all this information in one form.

Using the Add Table Wizard

After you have created or opened a project, you can use the Add Table Wizard to add new tables to add a detail table to an existing master table.

This section describes the Add Table Wizard. The following tasks are described here:

- Selecting database tables and columns
- Choosing visual components
- Defining additional components

For more information about working with master-detail information, see "Working with Master-Detail definitions" on page 3-28 for more information.

To start the Add Table Wizard:

1. Open the form to which you want to add the table.
2. From the Database menu, select Add Database Table.

   The Introduction page of the Add Table Wizard appears:
3. Click Next to identify a datasource.

Identifying a datasource

The Choose a datasource wizard page allows you to use a datasource that's already connected. You can also define a new database URL.

For information about adding datasources to your project, see "Identifying the datasource" on page 3-10.

- Click Next to choose available tables for your project.

Selecting tables

Specify the tables to use in your program on the Choose a Table wizard page. Select the table that contains the columns you want to work with. For more information see "Selecting tables or stored procedures" on page 3-13.

- Click Next to select columns from the specified table(s).

Selecting columns

The Choose Columns wizard page lets you select the database columns from the table you just specified. The information that the column contains can then be bound to visual components for display. For more information on the Choose Columns wizard page, see "Choosing database columns" on page 3-14.

- Click Next to select components for your form.

Selecting visual components

Visual components allow you to see and manipulate the data on your form. The Choose Components wizard page lets you select a component and specify a label for each database column. You can specify that the columns appear in a single grid. For more information on selecting components, see "Selecting visual components" on page 3-16.

- Click Next to select master-detail and join definitions.

Selecting master-detail and join definitions

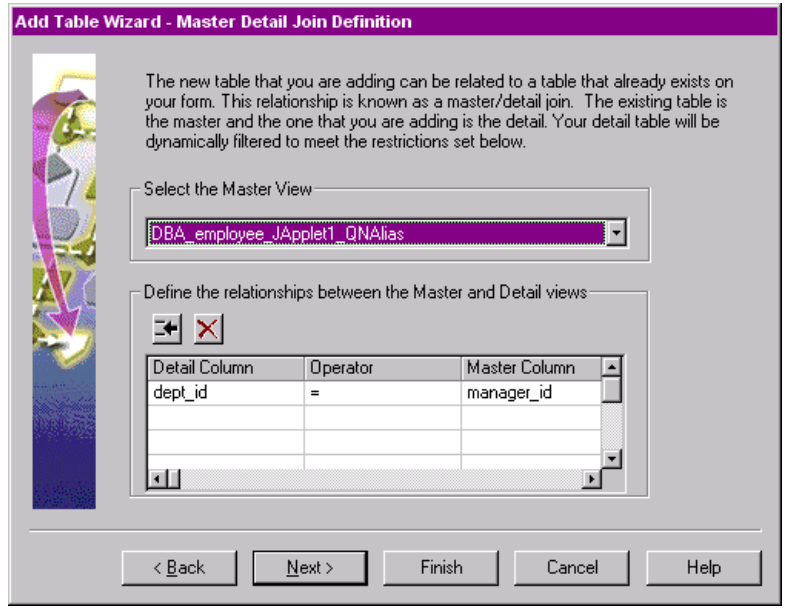
The Master-Detail Join Definition wizard page lets you select a table for a master view in a master-detail relationship and then create the join between information in the master column and the detail column according to the operator you specify.

The existing table you choose in this wizard page is the **master view,** and the table you're adding or creating is the **detail view**.

The list box displays the tables that represent `Query Navigator` components that exist in your project.

To create a master-detail relationship:

1. Choose a column for the master view. Click a cell under Master Column and select a column from the drop-down list that appears.
2. Choose a column for the detail view. Click a cell under Detail Column and select a column from the drop-down list that appears.
3. Choose an operator to define the relationship between the two columns. Click a cell under Operator and select a column from the drop-down list that appears.
4. To insert a new line between definitions, click Insert. To delete a line, click Delete.



5. Click Next to select additional components and layout options.

Note: You can also create a master-detail relationship by clicking the ellipsis icon on the right side of the `Join Columns` property of the `Query Navigator` component in the Property List. For more information on working with master-detail relationships, see "Working with Master-Detail definitions" on page 3-28.

Selecting additional layouts

The Additional Layout Options wizard page lets you select additional components for your applet or application. Using this wizard page, you can add a `Toolbar` and a `StatusBar`.

- Click Next to review your selections and finish this wizard.

Reviewing your selections

You can review your selections on the Summary page before adding them to your project. Click Back if you need to make changes in any of the previous pages. Otherwise, finish the wizard and add the table to your project.

To add the table to your project:

- Click Finish.

  The table is added and the Add Table Wizard closes.

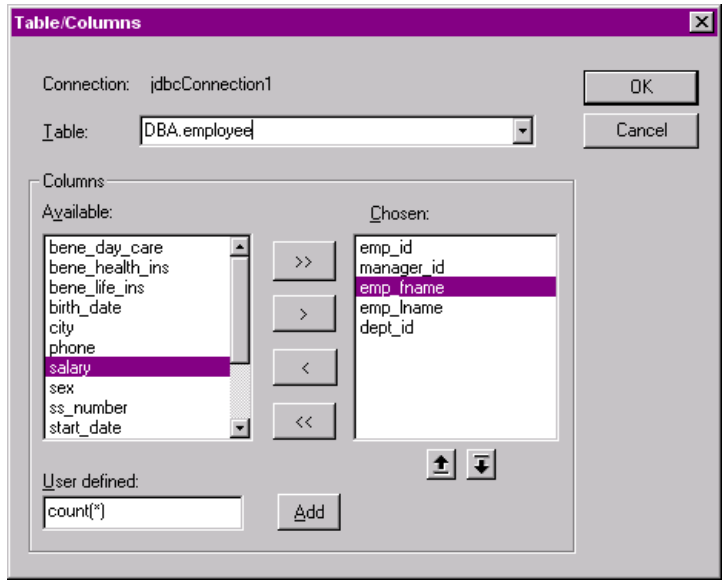
Using the Table/Columns Editor

Another way to add tables and columns to your databound project is by using the Table and Columns Editor. You access the Table and Columns Editor from the Property List.

To open the Table and Columns Editor:

1. In the Property List, select the record with which you want to work.
2. Click the `Table and Columns` property field.
3. Click on the ellipsis that appears to the right of the field.

The Tables/Column Editor appears:



4 Select the appropriate table from the Table drop-down menu.

The views for Available and Chosen populate.

5 Add or deselect columns by clicking the appropriate directional arrow.

6 If necessary, Reorder the columns in the Chosen view by clicking Up or Down.

7 If needed, you can type in a function in the User Defined text field. For example, by adding `count(*)`, you can count all the records in a database.

8 Click Add if you want the results of the function you added in the User Defined text field to appear as an additional selected column.

Working with Master-Detail definitions

A **master-detail relationship** is one where a record in one table, the **master**, can have a set of associated records in another table, the **detail**.

When you scroll through a master record, the application must display a set of detail records based on the new master record's values. Master-detail relationships restrict and filter the data automatically.

It's often useful to access data from two different tables in a master-detail relationship. When you create several smaller, related tables and link them together, you can drastically reduce the amount of redundant data, thereby making your program more efficient with accurate data.

For example, you may want to build a form that displays information about a department in a company and lists the employees in that department. In this case, the records in the employee table would have a **foreign key** (`dept_num`) that corresponds to the **primary key** (`dept_num`) in the department table. The `Query Navigator` and `Record Definition` components together represent a **view**.

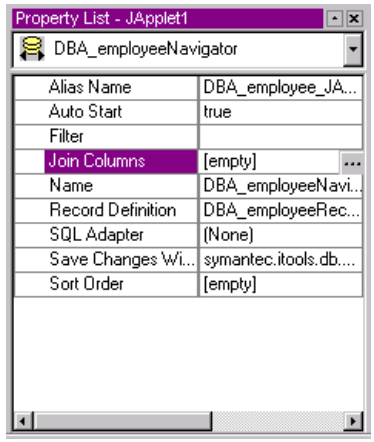
With the Master Detail Join Definition page of the Add Table Wizard, you can select a table as a master view and define a relationship, or **join** between a master column and a detail column. First, create a form with the master table. Then, add the detail table to it. See "Creating Joins with the Property List" on page 3-32 for more information on this step.

Creating or modifying a view

If your project already has a `Query Navigator` component, you can create a master-detail relationship, or modify an existing one. You can use the DataBound Project, Add Table Wizard, or the Master-Detail Editor to add a view to your project. The table you choose in the editor or in the wizard pages is the master view; the table you are creating is the detail view.

To create a view by using the Master-Detail Editor

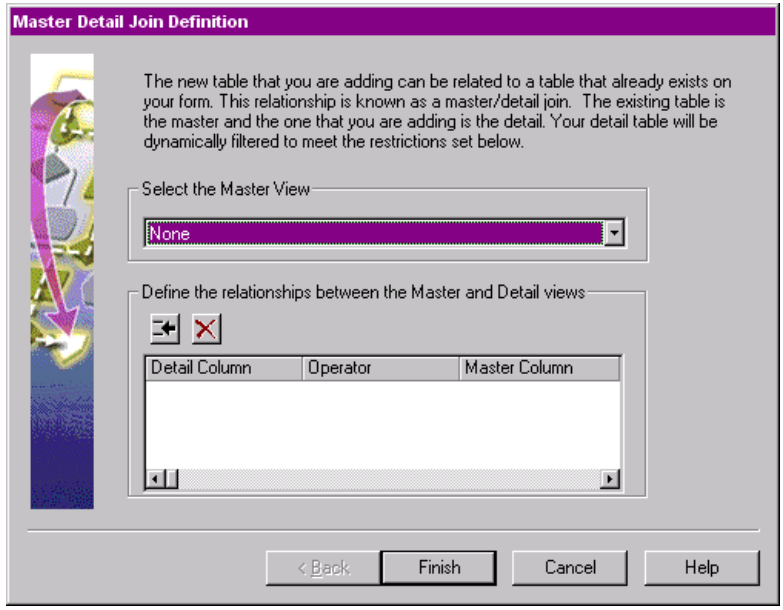
1. Select the `Query Navigator` component in your project that contains the view you want to use. You can select the `Query Navigator` in the Form Designer or the Property List.
2. Select the `Join Columns` property in the Property List.



Ellipsis appear to the right of the field.

Click the ellipsis to open the Master-Detail Editor.

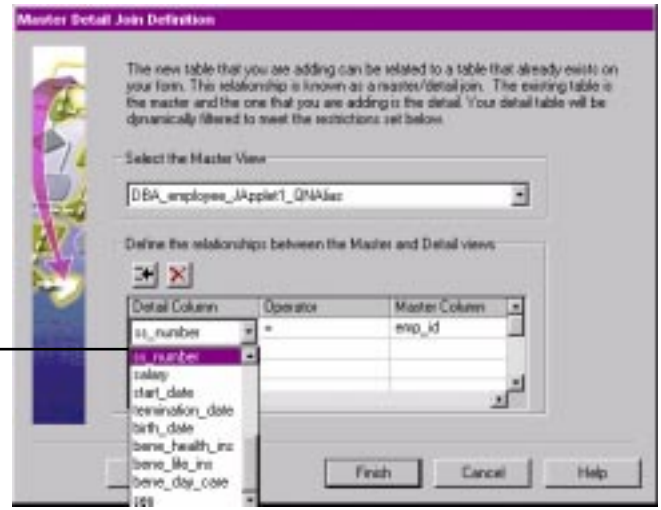
The Master-Detail Editor displays:



3 Use the Select the Master View drop-down list to specify an available master view.

4 Click a cell under Detail Column and select a detail column from the drop-down menu:

Select items from the drop-down menus.



5 Click an Operator field and select an operator from the drop-down menu.

6 Click the Master Column field and select a master column from the drop-down menu.

7 Click Finish to create or edit the joins.

To create a master-detail relationship with the DataBound Project wizard:

- Use the DataBound Project Wizard to create a master-detail relationship while creating a new project. See "Starting the DataBound Project Wizard" on page 3-5.

To create a master-detail relationship with the Add Table Wizard:

1 In the Project window, select the container that you want to add the table to, then from the Database menu, select Add Database Table.

2 Complete all the pages in the wizard. For more information, see "Adding database tables" on page 3-23.

To add a master-detail relationship table using the Add Table Wizard:

1 Open the form to which you want to add the table.

2 Choose Add Table Wizard from the Database menu to launch the Add Table Wizard.

3 Complete the Add Table Wizard as described in "Adding database tables" on page 3-23.

Creating Joins with the Property List

A master-detail relationship is the relationship between a master view and its detail view(s). You specify this relationship by setting the join properties in a `Query Navigator` component.

To create a detail view by changing the Join properties

1 In the Project Window, select the `Query Navigator` component to represent the detail view.

2 In the Property List, examine the `Join` property field.

3 Select the master `Query Navigator` in the `Alias Name` field.

Note: The `Alias Name` is a property in the Property List; it might not match the object name. You can enter a name that does not exist yet. The `Query Navigator` you enter does not have to be contained by the same form as the component.

---

4 Click the `Join Columns` property field.

5 Click the ellipsis to open the Master-Detail Join Definition Editor.

6 Use the Master-detail Join Definition Editor to specify the join relationship.

Note: The Master-Detail Join Definition Editor overrides the SQL statement created for the detail view, depending on the join criteria that is specified.

Handling changed detail records

If you make changes in the detail result set, and then scroll the master record, Visual Cafe displays a dialog box asking you whether to save the changes, discard the changes, or cancel the scroll operation. If you choose to save the changes, then all changed (dirty) records will be saved for all result sets in the relationship, including the master. That is, the dialog issues a call to save records with the `Save All Levels` method to ensure data integrity.

Changing views

When you change or add a detail record such that it does not match the current detail set, the record appears in the cached detail set until it is saved. Saving the record will update it in the database, after which it will be removed from the cached result set. However, this is only the case if the join operator is equal (`=`) or not equal (`!=`). It is not removed for all other operators, such as less than (`<`).

If you set the `Auto Start` property, in the Property List of a detail `Query Navigator` to `false`, you need to start the `Query Navigator` programmatically before it will perform any operations.

To change the data model for a view:

- Change the `Table` property for the `Record Definition` component in the Property List.

To change the relationship between records in a view:

- Change the `Filter` property for the `Query Navigator` component in the Property List.

To change a detail view by changing the Join properties:

1. In the Project window, select the `Query Navigator` component that represents the detail view.
2. In the Property List, examine the `Join` properties.
3. If needed, specify the master `Query Navigator` in the `Alias Name` field in the Property List.
4. Click the `Join Columns` property field.
5.  Click the ellipse button to open the Join Definition Editor.
6. Use the dialog box to change the join definition.

To delete a view:

- Delete the `Query Navigator` component. You can delete the `Record Definition` component as well, if you are not using it with another `Query Navigator` component.

Using dbNAVIGATOR in form development

The dbNAVIGATOR datasource browser lets you access databases. The dbNAVIGATOR provides a hierarchical view of datasource cataloging information known as metadata.

You can use the dbNAVIGATOR to quickly perform many database functions by dragging specific database fields and columns from the dbNAVIGATOR window and dropping them onto your form in the Form Designer.

This section describes:

- Viewing database resources
- Using the dbNAVIGATOR drop-down menu
- Adding servers
- Viewing datasources

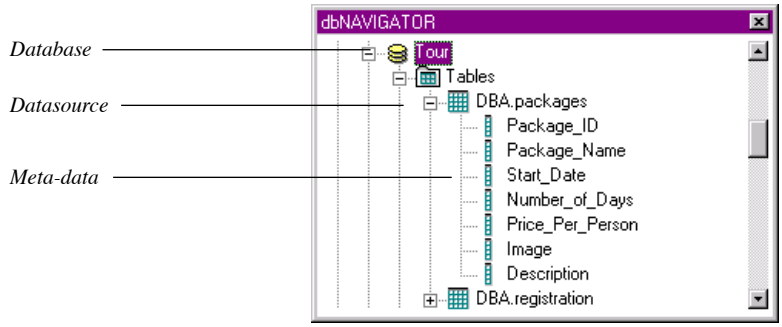
Viewing database resources

When you connect to a database server, dbNAVIGATOR displays a list of datasources, such as servers, that are currently connected, as well the tables and columns in each datasource.

To display the dbNAVIGATOR:

- From the View menu, select dbNAVIGATOR.

  If you've already connected to a datasource from Visual Cafe, the dbNAVIGATOR window is displayed. Otherwise, you're prompted to provide some information for the server you want to access. For more information, see "Defining a datasource" on page 3-2.



The dbNAVIGATOR displays a hierarchy with the following items (in order from highest in the hierarchy to lowest):

Item	Description
Server items	Represents a server that can be accessed by the computer that Visual Cafe Database Edition is running on. You can't drag a Server item to a form in a project.
Datasource items	Represents a database that's connected to the associated database server. Dragging a datasource item to a form in a project creates `Connection Manager` and `JdbcConnection` objects, which manage connections to a datasource. You have one `Connection Manager` object per project, and one `JdbcConnection` object per datasource.
Database table items	Represents a table in the associated database. When you connect to a database, dbNAVIGATOR displays the tables and columns that are in the database. Dragging a database table item to a form in a project creates a top-level `Record Definition` object and a `Query Navigator` object, which is contained by the form.
Database column items	Represents a column in the associated database table. Dragging a database column item to a project creates a databound component. You can specify the default component type by choosing Environment Options from the Tools menu, then clicking the associated Database tab.

When you drag an item that's low in the hierarchy into a project, the associated items higher in the hierarchy are added automatically to the project for you.

Within the dbNAVIGATOR window, you can expand and contract the items in the hierarchy. If you expand a server item, you see the datasource items on the server. If you expand a datasource item, you see the database table items in the datasource. When you expand a database table item, you see the database column items in the table.

Building databound forms

Once you have access to a datasource through dbNAVIGATOR, you can drag items from dbNAVIGATOR and drop them directly onto your form. Some itmes you can move this way are table and columns.

Database table items in dbNAVIGATOR

A database table icon represents a table in the database. When you connect to a database, dbNAVIGATOR displays the tables and table columns that are in the database. You can add a database table to a form in your project from dbNAVIGATOR.

To add a table to a form using dbNAVIGATOR:

- Select the table you want to add from the dbNAVIGATOR window and drag it onto the open form.

Database column items in dbNAVIGATOR

A database column item represents a column in the database table. When you connect to a database, dbNAVIGATOR displays the tables in the database and the columns in the tables. You can add a column from a table to a form in your project.

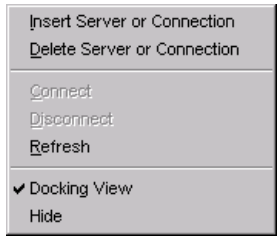
To add a column to a form using dbNAVIGATOR:

1. Open the table from the dbNAVIGATOR window that contains the column you want to add to the open form.
2. Select and drag the column from dbNAVIGATOR and drop it onto the open form.

   The column displays in the form.

Using the dbNAVIGATOR pop-up menu

When you right-click in the dbNAVIGATOR, a pop-up menu displays:



The pop-up menu presents you with the following options:

- Insert Server or Connection
  Opens the Insert datasource dialog box, which lets you connect to a server or datasource.
- Delete Server or Connection
  Deletes the selected server from the dbNAVIGATOR window.
- Connect
  Opens the Logon dialog box, which lets you log on to a datasource.
- Disconnect
  Disconnects the selected database. This command is available only if a datasource item is connected.
- Refresh
  Refreshes the information in the selected databases' dbNAVIGATOR tables and columns. This command is useful for multiple concurrent users.
- Docking View
  Toggles between docked or floating dbNAVIGATOR window.
- Hide/Close
  Hide is available when Docking View is selected, Close is available when dbNAVIGATOR is floating. Both close the dbNAVIGATOR window.

To add a new datasource in dbNAVIGATOR:

- Right-click in the dbNAVIGATOR window and choose Insert Server or Connection from the pop-up menu.

  The Insert a datasource wizard page appears. For more information, see "Inserting a datasource" on page 3-11.

To add an existing datasource in dbNAVIGATOR:

1. Right-click in the dbNAVIGATOR window and choose Insert Server or Connection from the pop-up menu.

   The Insert a Datasource wizard page appears.

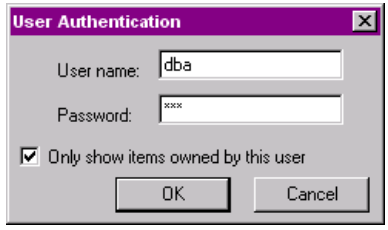
2. Click Connect to a Defined datasource.

   The list of available database URLs becomes active.

3. Select the appropriate datasource.

4. Click Next.

   The User Authentication dialog box appears, if required by the datasource:



5. Log on to the datasource with the correct user name and password in the User Authentication dialog box.

6. Click OK to submit the logon request.

   The Insert a datasource page closes.

To view datasources in dbNAVIGATOR:

1. Choose dbNAVIGATOR from the View menu to display dbNAVIGATOR.
2. Click + beside a server to show the datasources registered to that server.
3. Click + beside a datasource to display its tables.

   If you have not yet logged on to this datasource, you're prompted for a user name and password. Enter that information and click OK.
4. Click + beside a table to show its columns.

Refreshing the dbNAVIGATOR window

Refreshing the dbNAVIGATOR window updates the display of the tables and columns in a datasource to reflect changes in the database. Refreshing the dbNAVIGATOR window is useful, for example, when there are multiple concurrent users who are making changes to the database at the same time.

1. From the Window menu, choose dbNAVIGATOR.
2. Right-click a server or datasource item, then choose Refresh from the pop-up menu.

   dbNAVIGATOR updates itself and redisplays.

Using more than one datasource

In medium- and large-scale database projects, you may want to use data from more than one datasource and/or from more than one server.

When you drag an item that's low in dbNAVIGATOR's hierarchy from dbNAVIGATOR into a project, the necessary items higher in the hierarchy are added to the project for you. For more information, see "Viewing database resources" on page 3-34.

Here's some useful information about components you'll need if you're using more than one datasource:

- `Connection Manager` and `Jdbc Connection` objects manage connections to a datasource. You have one `Connection Manager` object per project, and one `JdbcConnection` object per datasource. To add these objects, drag a datasource item to a form in a project.
- You need a top-level `Record Definition` object and a `Query Navigator` object, which is contained by the form, for each database table. To add these objects, drag a database table item to a form in a project.
- A single `Record Definition` object can be used by multiple `Query Navigator` objects. The `Record Definition` object defines the structure of a table and handles data manipulation, such as inserting rows, updating rows, and so on. The `Query Navigator` objects define which rows of data will be retrieved from the database, how they will be inserted, and so on.

For more information about these components, see "Datasource components" on page 4-13.

Using the Data Connection Navigator

The **Data Connection Navigator** (dcNAVIGATOR) is a component that contains all the current data connection objects. It has the same functionality as the Data Binding Editor, but it displays connection objects as a tree. You can use the dcNAVIGATOR to select data connection objects that might be difficult to find visually.

The dcNAVIGATOR can be accessed when a component requires a datasource: Data Binding Editor, `Binder Model`, `Calculation` or `Validation Adapters`, or `Stored Procedure Adapters`.

To access dcNAVIGATOR:

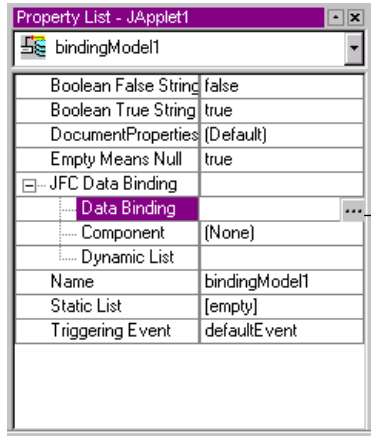
1 You project must have a `BindingModel` component and a Swing component in it.

1 In the Property List, select `BindingModel` component.

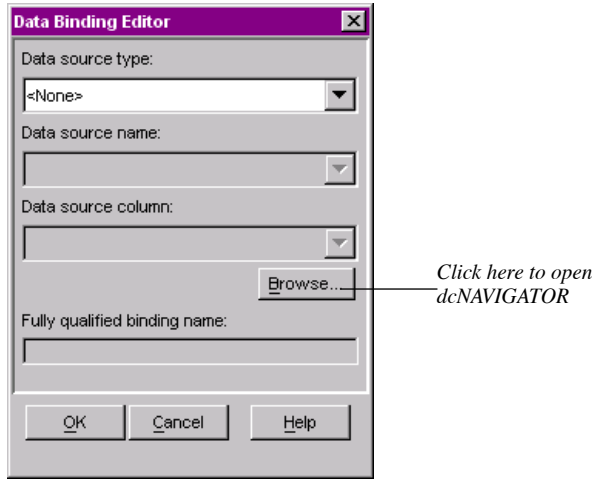
The properties for that `BindingModel` appear.

2 Select `Data Binding` property of the Swing data binding component.

3 Click the ellipsis that appear to the right of the property field to open the Data Binding Editor:

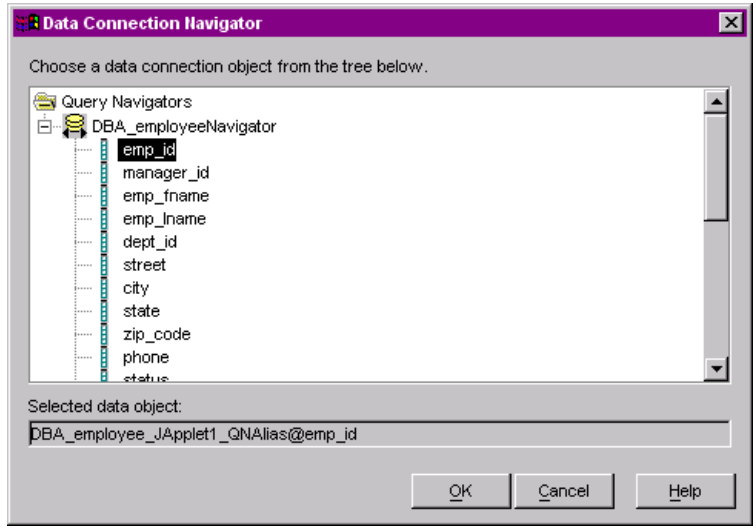


The Data Binding Editor appears:



4 Click Browse to open the dcNAVIGATOR window.

The dcNAVIGATOR window appears:



To add an object from the dcNAVIGATOR to your Project

1 Expand any of the folders by clicking + until you find the desired data connection object.

2 Select the desired data connection object.

The fully qualified binding name of the data connection object appears in the Selected data object text field.

3 Click OK to close dcNAVIGATOR and return to the Data Binding Editor.

The Data Binding Editor populates the text fields from your selection in the dcNAVIGATOR window.

4 Click OK to apply your selections and close the Data Binding Editor.

Disconnecting from a datasource

You can disconnect from a datasource while in the dbNAVIGATOR window. Disconnecting from a datasource is useful if you need to free a connection to a server that has a limited number of connections.

To disconnect from a datasource:

1. Choose dbNAVIGATOR from the View menu.
2. Expand the server item (click +).

   The datasources on the server appear.
3. Right-click the datasource item, then choose Disconnect from the pop-up menu.

Working with the Grid component

The `Grid` component allows you to display multiple data records in an *array*, or spreadsheet-like display. Once you've added a grid to your project, you can customize it in various ways:

- adjust column widths
- change the names of column headings
- add predefined buttons to the bottom of the grid
- change attributes of grid cells and columns
- enhance the Grid Toolbar

Adding a Grid component to your project

The `Grid` component is available from the AWT DataBound tab of the Component Palette. When you drop a `Grid` component on a form, the `Grid` is empty. You won't see data populated in the `Grid` until run time.

The easiest way to add and populate a `Grid` component is to drag a table name from dbNAVIGATOR onto a form. See "Using dbNAVIGATOR in form development" on page 3-34. Visual Cafe Database Edition creates the `Query Navigator` component and fills in its properties. Then, you select the `Grid` component from the AWT DataBound tab on the Component Palette, drag it to the Form Designer, and size it as needed. Last, set the

`Grid`'s data binding properties through the Data Binding Editor, which can be accessed from the Property List.

To add a Grid component using dbNAVIGATOR:

1 Drag the table icon from the dbNAVIGATOR window and drop it onto the open form. For information on using dbNAVIGATOR, see "Using dbNAVIGATOR in form development" on page 3-34.

Visual Cafe automatically creates all the other components that are needed to define the database connection, such as the `Connection Manager` and `JdbcConnection` components.

The columns display in the Grid, from left to right, in the same order that they appear in the dbNAVIGATOR window.

2 Drag the `Grid` component from the Component Library or AWT DataBound tab of the Component Palette to add it to your form.

The `Grid` is empty. You won't see the actual data until run time.

3 Make the `Grid` the size you want by dragging the bottom right corner.

If columns do not fit within the `Grid` component, a scroll slider appears.

4 Adjust the column widths by positioning the cursor on the column heading until a double-edge arrow displays. Then drag the column border line left or right.

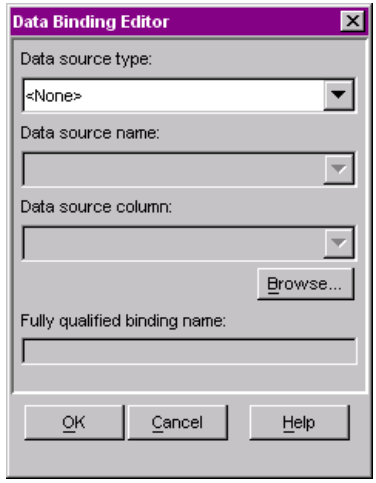
Note: If you want to edit the text in a column heading, you must do so in the source code.

To populate a Grid component:

1 Set the `Grid` component's `Data Binding` property in the Property List.

2 Click on the ellipsis that appear to the right of the property field to open the Data Binding Editor.

The Data Binding Editor appears:



3 In the Data Binding Editor, select a `Query Navigator`.

4 Select the column data to populate the Grid from the Available Columns view.

The fully qualified binding name for each component appears in the text field at the bottom of the Data Binding Editor.

5 If necessary, reorder the columns by clicking Up or Down.

6 Click the Limit number of displayed rows checkbox if you want to limit the number of rows that are displayed. The default number is 10.

7 Click OK to apply the selections and close the Data Binding Editor.

Note: `Alias Name` is a property in the Property List; it might not match the object name. You can enter a name that does not exist yet. The `Query Navigator` that was created does not have to be contained by the same form as the component.

---

8 To run the form, choose Execute from the Project menu.

The `Grid` component uses the database column names as the `Grid` column headings.

You can narrow the query by adjusting the `SELECT` statement. For more information, see "Defining SQL SELECT statements through properties" on page 5-2.

Changing grid cell attributes

You may want to change the visual attributes, such as color, font, and row highlight color, of grid cells. To change these attributes, you modify the `Grid` component's source code. Place your custom source code after the `INIT_CONTROLS` code block.

Setting foreground and background colors

The Property List does not provide a property to control the foreground and background colors of grid cells. You must specify these attributes in the source code.

Following is a code sample that changes a cell's background and foreground colors. The `setCellBgColor` and `setCellFgColor` methods take as arguments a row number, a column number, and a new color:

```
grid1.setCellBgColor(1, 1, Color.blue);
grid1.setCellFgColor(1, 1, Color.white);
```

Note: Cell row and cell column numbers start with one, not zero.

You can also change the foreground and background colors of an entire row or column of cells. The method arguments are a row or column, number, and a color:

```
grid1.setColBgColor(1, Color.blue);

grid1.setColFgColor(1, Color.white);

grid1.setRowBgColor(1, Color.red);

grid1.setRowFgColor(1, Color.white);
```

Changing fonts

You may want to change the font of a cell or range of cells for emphasis. The following examples illustrate how to change cell, column, and row fonts:

```
grid1.setCelllFont(2, 2, new Font ("Helvetica",
Font.PLAIN, 14));

grid1.setColFont(1, new Font ("TimesRoman",
Font.ITALIC, 18);

grid1.setRowFont(1, new Font ("Courier", Font.BOLD,
16)
```

The arguments are the column number (columns are one-based, not zero-based) and a color as defined in the `java.awt.Color` class.

Changing grid column attributes

You can change additional attributes of grid columns, such as heading names, column alignments, heading colors, and heading fonts, by modifying the `Grid` component's source code. Place your custom source code after the `INIT_CONTROLS` code block.

Changing column headings

You can change the text in column headings by adding custom code, as follows:

```
grid1.setHeading ("Last Name", 1, 15);

grid1.setHeading ("First Name", 2, 15);
```

The third `setHeading` argument specifies the width of the column in average character widths.

Changing the column alignment

You can also modify column alignment. The method arguments are the column number and the alignment type. The alignment can be specified by static final `int` values attached to the `Grid` class:

```
grid1.setColumnAlignment (1, grid1.LEFT);

grid1.setColumnAlignment (2, grid1.CENTER);
```

```
grid1.setColumnAlignment (3, grid1.RIGHT);
```

Changing column heading colors and fonts

You can modify column heading and font attributes. The `setHeadingColor` method changes the foreground and background colors of a column heading. The `setHeadingFont` method changes the font of a column heading. The method arguments for the `setHeadingFont` method are font name, font face, and column number. The arguments for the `setHeadingColors` method are the column number and color.

```
grid1.setHeadingColors (1, Color.black, Color.white);
```

```
grid1.setHeadingFont (new Font("TimesRoman",
Font.BOLD, 18), 1);
```

Defining automatic Grid redraw

Each time a `Grid` component is updated, it is automatically redrawn. If you're going to make several changes to your `Grid` (column headings, cell color changes, and so on) it's a good idea to turn off the automatic redraw capability. Once you've made all your changes, you can turn it back on:

```
grid1.setAutoRedraw (false);  // turns off auto-redraw
   // make changes to the grid here
grid1.setAutoRedraw (true);   // turns on auto-redraw
```

Defining automatic Grid row numbering

You can turn automatic row numbering on or off. When automatic row numbering is activated, the starting row number can be set by using the `setupAutonumbering` method, for example:

```
grid1.setupAutonumbering (1);  // begins auto-row
numbering at 1
```

```
grid1.setupAutonumbering (5);  // begins auto-row
numbering at 4
```

```
grid1.setupAutonumbering (0);  // turns off auto-row
numbering
```

Modifying the Grid toolbar

You can add functionality to a `Grid` toolbar by:

- creating a customized `Grid` event handler with the added functionality
- then informing the `Grid` about the new event handler by using the `installEventHandler`

Each `Grid` component has a `DefaultTvEventHandler` object that provides the standard functionality of the `Grid`: Insert, Go To, Undo, Restart, Delete, Undelete, and Save. Additional functions are made through a custom `EventHandler` object that extends from the `DefaultTvEventHandler`.

Adding search capability

You can add a few components to the toolbar and implement a standard search capability within a column. These features include a search field, a forward and backward search direction radio button, and a Go button (to start the search).

Sample source code is available in the Online Help. To access this source code, press F1, then use the Index or Search tools to find "Modifying the Grid Toolbar."

To add search capability to the Grid Toolbar:

1. Create a project with a `Grid` component. See "Selecting visual components" on page 3-16 for more information.
2. When the `Grid` is functional and displays data, add import statements at the beginning of the `Grid` source code.
3. Add the sample code from the Online Help for a custom event handler. Code notes follow the code sample.

Once you've constructed the new `MyEventHandler` object, you must connect it to the existing `Grid` object. Use the `installEventHandler` method with an instance of the `MyEventHandler` object as a parameter, as shown in the sample code. Be sure to read the notes for this section of code.

Protecting a Grid column

You can protect cells, so that the user cannot modify the data, by enhancing the component's Java code. Like Grid cell colors, you can protect specific cells, a whole row, or a whole column. Place any custom source code after the `INIT_CONTROLS` code block.

Here are some examples:

```
grid1.setCellEditable(1, 1, false);

grid1.setColEditable(2, false);

grid1.setRowEditable(2, false);
```

In the above syntax, the first argument is the column number and the second argument is Boolean, where false is read-only and true is read-write.

Calculation and validation rules

Visual Cafe Database Edition allows you to support **calculation rules** and **validation rules** in your projects. These rules are also known more generally as business rules. Two adapters make it possible for to implement this support:

- `Calculation Adapter`
- `Validation Adapter`

`Calculation` and `Validation Adapter`s allow you to use any of the predefined calculation and validation rules that are included with Visual Cafe. You can modify the rules and adapt them to your needs or make them available to the users of your programs. You can also import custom rules that exist outside of the Visual Cafe environment. You can find these adapters in the Data Source tab of the Component Palette.

There are two steps to incorporating rules in Visual Cafe. The first step is to develop a calculation rule as a separate Java class by implementing the specified `Calculation Column` or `Validated Column` interfaces. For examples of creating these rules, see Appendix , “Creating Custom Calculation and Validation Rules,”.

Once the rule is developed, the second step is to integrate the rule with the database applet or application and invoke the rule for different database events. When you want to create a calculation for a column or validate data, you can use the appropriate adapter by dragging and dropping it into the project. Then, you customize the adapter and add it to your project. Any AWT DataBound or Swing component can listen for this calculated or validated data and then display the output or pass it to another object.

Both adapters work with `Query Navigator`, which accept input from users. When input to a particular rule changes, the `Validation Adapter` applies the formula and sends the output to other databound objects that request it.

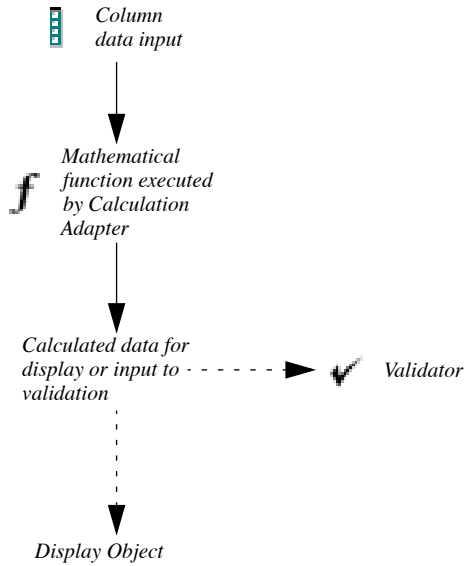
Note: Don’t mix database columns from different `Query Navigator`s in the same adapter.

Visual Cafe supports only client-side rules; that is, all calculations and validations are performed on the client side. However, you can integrate server-side rules by invoking them from the client.

About Calculation Adapters

You use a `Calculation Adapter` to specify a calculated column. This adapter performs the calculations when it receives new input. You can add a `Calculation Adapter` to your project, then customize it with the Add Calculated Data Wizard. See "Using calculated data" on page 3-53.

The following diagram illustrates how a `Calculation Adapter` works.



In the above illustration, the `Calculation Adaptor` performs all the necessary calculations and then publishes them as the total of that operation. The resulting calculation can also be passed to a `Validation Adapter`. For more information, see "Validating data" on page 3-60.

Using calculated data

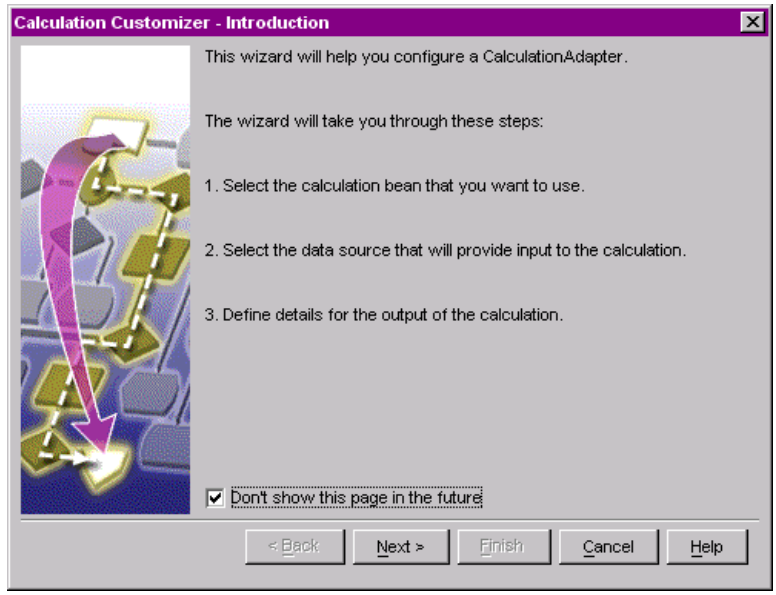
The Calculation Customizer helps you customize a `Calculation Adapter` that will perform mathematical functions on a column of data. Since the `Calculation Adapter` contains some of the most commonly used mathematical functions, you only need to provide the name of a datasource and the columns to be calculated. In addition, you can import custom calculation rules with the customizer.

Before customizing a `Calculation Adapter`, you must map all database columns to AWT or Swing components by using the Add Table Wizard. For more information, see "Adding database tables" on page 3-23.

To start the Calculation Customizer:

1 From the Database menu, select Add Calculated Data.

The introduction page of the Calculation Customizer appears:



2 Click Next to select a calculation.

You can also use drag and drop techniques to start the Calculation Customizer.

To start the Add Calculated Data Wizard using drag and drop:

1 Drag a `Calculation Adapter` from the Data Source Toolbar onto the Form Designer.

2 Right-click on the `Calculation Adapter` icon and select Customize from the pop-up menu.

The Calculation Customizer appears.

3 Click Next to select a calculation.

Choosing a calculation

Use the Choose Calculation page to select the function that you want to apply to your data. Some of the many predefined functions are: Min, Max, Multiplication, Subtraction, Addition, Division, and String Concatenation.

To select a calculation:

1. Click the Calculation Formulas folder of the Component Library field.

   A list of the available functions display.

2. Click the function you want to use.

   The name of the selected calculation appears at the bottom of the page.

Alternatively, you can import a function from another datasource.

To import a calculation function:

1. In the Choose Calculation page, click Import a custom calculation.
2. Click Import.

   An Open File dialog box displays allowing you to select the file that contains your function.

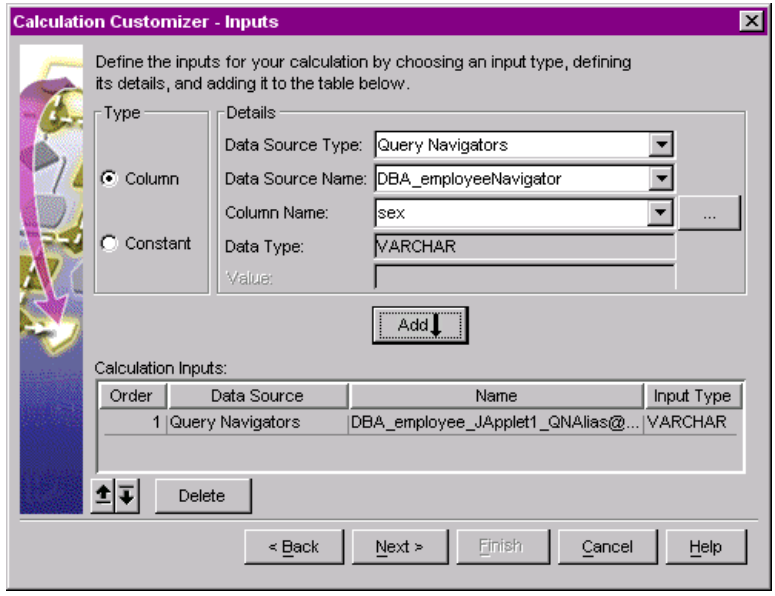
3. Select the file that contains your function and click Open.

   The name of the custom calculation appears in the Selection dialog box at the bottom of the Choose Calculation page. If you want to reuse this custom rule in other projects, click Add custom rule to Component Library.

4. Click Next to define input parameters for the `Calculation Adapter`.

Defining input parameters

The Inputs page of the Calculation Customizer allows you to set the input parameters for the `Calculation Adapter`:



There are two types of input: data from a datasource, or a constant. When you use data from a datasource, select Column. When you want to use a constant (unchanging) number or string, select Constant. Different details display for each type of input you select.

If Column is selected, details that you can change include datasource type, datasource name, and column name. A data type is passed from the datasource to the `Calculation Adapter` and should not be changed.

If Constant is selected, details that you can change include data type and value. You must select a data type for your value.

Change any necessary fields for your input type and click Add to add your input parameters to the adapter. You can add multiple input parameters to a `Calculation Adapter`.

To define input parameters:

1. Select a Column or Constant type for the input parameter. Column is selected as the default input parameter type.
2. In the Details box of the Input page, set the following items:

Input details	Description
Column	When you use data from a datasource, select Column. A data type is passed from the datasource to the `Calculation Adapter` and should not be changed
Constant	When you want to use a constant, unchanging number or string, choose Constant. You must select a data type for the constant.
Datasource Type	The default selection is `Query Navigator`. If the input parameter for the `Calculation Adapter` depends on another datasource type, select it from the combo-box.
Datasource Name	When you select a datasource type, the datasource names that are registered to that datasource type appear.
Column Name	The available column names that are part of the defined datasource type and name appear in the combo-box.
Data Type	If you've selected Constant for the Input Type, the Data Type field becomes active. You can select a data type from the combo-box. The default data type is `INTEGER`.
Value	If you've selected Constant for the Input Type, the Value field becomes active. The value for this field must be of the type that you specified in the Data Type field.
Add	Click Add to add the input parameters to the `Calculation Adapter`.

Input details	Description
Delete	Click Delete to remove the input parameters from the `Calculation Adapter`.
Up or Down	You can click the Up or Down arrows to specify the order in which input parameters are processed.

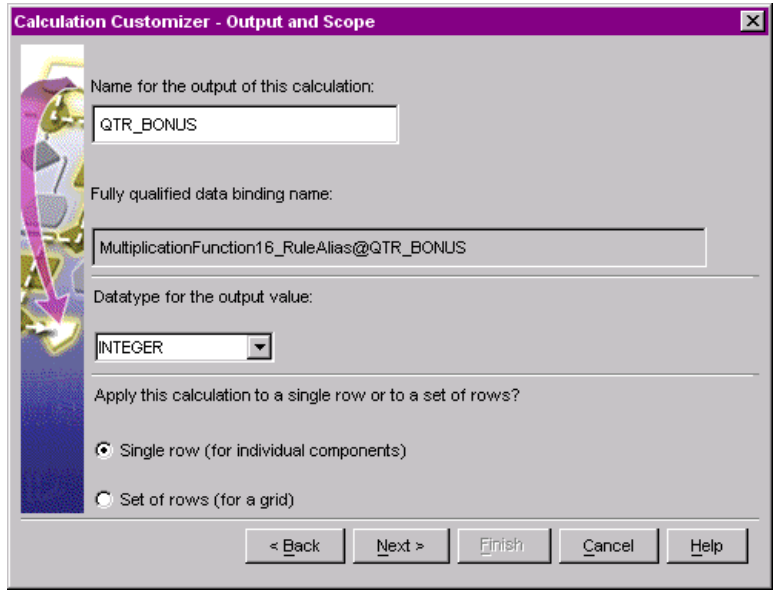
The input parameters display in the table at the bottom of the page.

3 Click Up or Down to reorder the input parameters.

4 Click Next to define output parameters.

Defining output parameters

Output is the result of the calculation the `Calculation Adapter` performs. The Output and Scope wizard page lets you define the output name, the data binding name, the data type of the value your `Calculation Adapter` will return, and how you want to apply the data across rows.



To create an output name:

- Type an output name in the text field.

To set the data type:

- Select the data type from the drop-down menu that matches the data type of the output parameter.

To apply a calculation to a single row:

- Select a single row.

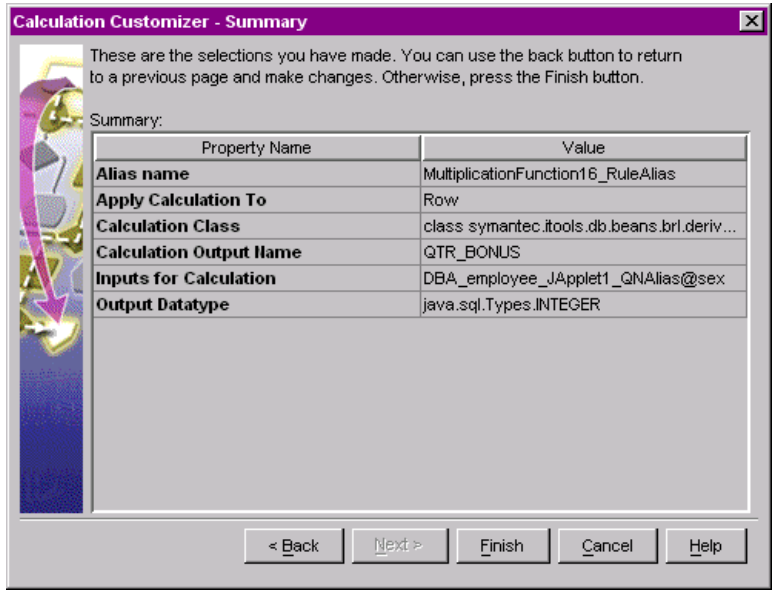
To apply a calculation to a set of rows:

- Select a set of rows.

  When you have specified the output parameters, click Next to review your selections.

Reviewing your selections

With the Summary page, you can review the selections you've made. If you are satisfied with your selections, click Finish to create the `Calculation Adapter` and close the Calculation Customizer.



Validating data

You use a `Validation Adapter` to compare changes in data that a user inputs against the data in a database. When data matches, the validation rule doesn't execute. Validation rules are performed on the client side. To see how `Validation Adapter`s work, consider the following example.

A user fills in a wine purchase order form. If the user enters a state name where buying alcohol from out of state is not allowed, the validation rule checks the stored information in the database. If the `Validation Adapter` notices an invalid entry that was entered, the user is informed that their entry isn't valid and no data is changed in the database.

Whenever source code is changed for any of the data items, the validation is executed. The `Validation Adapter` executes the Java class that contains the validation formula. The output of the validation is published as a Boolean value based on the outcome of the validation. You can implement a custom or default dialog to the users of your program notifying them to re-enter their data. You can also re-direct this message to the `Log` window.

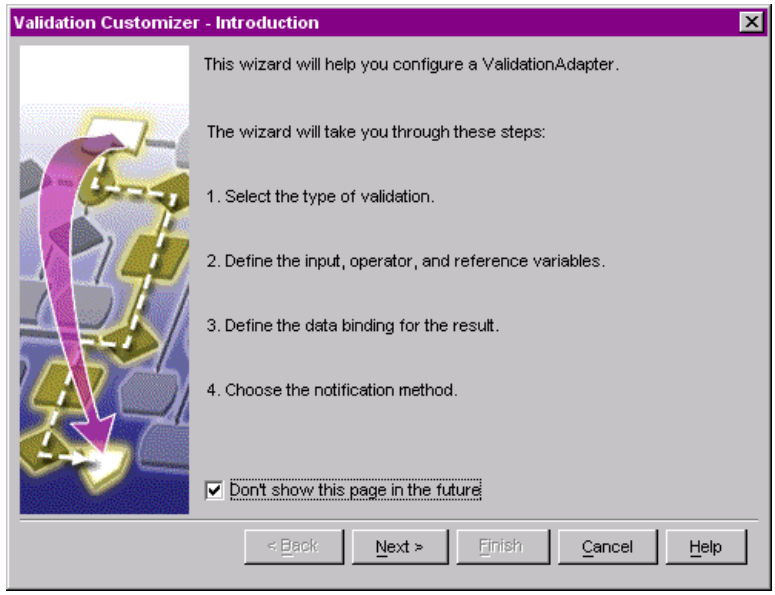
Starting the Validation Customizer

You can start the Validation Customizer from the Visual Cafe main menu, or by using drag-and-drop techniques. When you use the Visual Cafe main menu, the Validation Customizer adds the `Validation Adapter` to your project for you. If you use drag and drop, you'll invoke the Validation Customizer from a `Validation Adapter` component.

To open the Validation Customizer:

1 From the Database menu, select Add Validation.

The introduction page of the Validation Customizer appears:



2 Click Next to select or import a validation formula.

To open the Validation Customizer using drag and drop:

1 Drag a `Validation Adapter` from the Data Source tab of Component Palette and drop it onto the Form Designer.

2 Right-click on the `Validation Adapter` and select Customize.

The opening page of the Validation Customizer appears.

3 Click Next to select or import a validation formula.

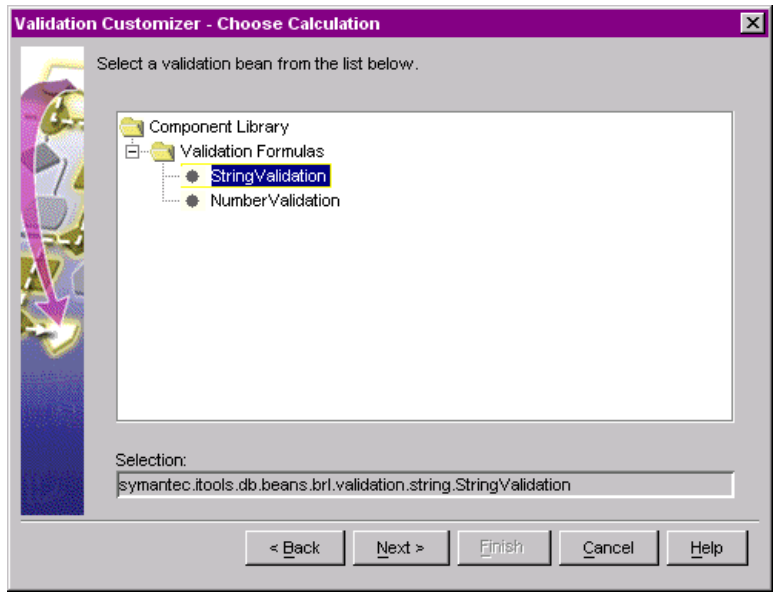
Selecting a validation formula

Use the Select a Validation Formula wizard page to select the validation rule type that you want to apply to your data. The available types of validation rules are `String` and `Number`.

To select a validation rule:

1 Click + that is next to the Validation Formulas folder in the Choose Calculation page of the wizard.

The available validation formulas appear:

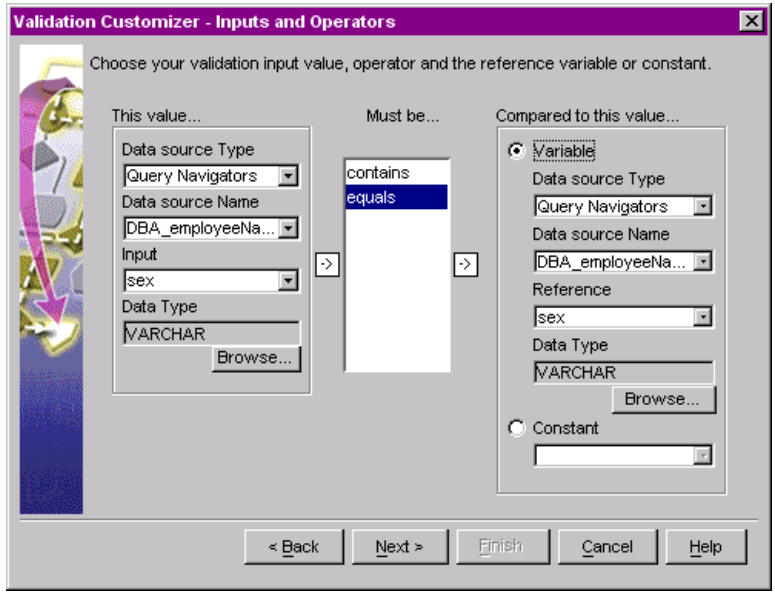


2 Select the validation formula you want to use.

3 Click Next to define input parameters and operators for the `Validation Adapter`.

Defining input parameters

The Inputs and Operators page allows you to set the input parameters and operators for your validation:



You'll need to select a validation input value, a validation operator, and the reference value you want the input value compared against.

Input values can come from a column in a table, a stored procedure result set, or the result of a `Calculation` or `Validation Adapter`. Operators allow you to set relationships between columns. The available values include greater than (`>`) , less than (`<`), equal (`=`) or not equal (`!=`) to each other.

There are two types of **reference values**: data from a datasource or a constant. When you use data from a datasource, choose Variable as the datasource type. When you want to use a constant, unchanging number or string, choose Constant. Different details display for each type of input you select.

If you select Variable, details that you can change include datasource type, datasource name, and a reference, which is a column name. A data type is passed from the datasource to the `Validation Adapter` and can't be changed.

If you select Constant, details that you can change include data type and value. You must select a data type for your value.

To select an input value:

1. Select the appropriate datasource type from the drop-down menu: `Query Navigator, SQL Adapter, Calculation Adapter`, or `Validation Adapter`.
2. Select the datasource name.
3. Select the input (column name) you are using for the input value.

   The data type for the input column appears as you select values from the drop-down boxes.

To select a validation operator:

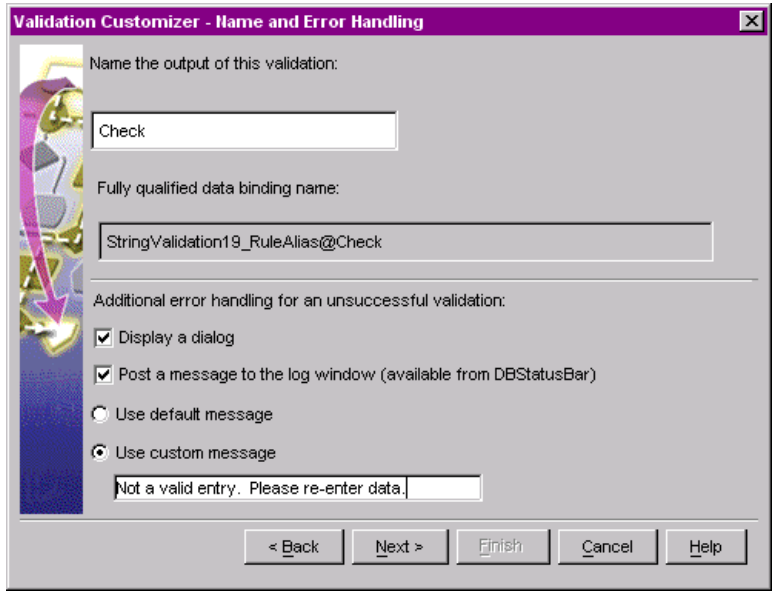
- Choose the appropriate operator to compare your input value against your reference value.

To select a reference value

1. Click Variable or Constant as appropriate. If you click Constant, enter a constant value in the Constant field.
2. Select the appropriate datasource type; `Query Navigator, SQL Adapter, Calculation Adapter`, or `Validation Adapter`.
3. Select the datasource name from the drop-down menu.
4. Select the reference value (column name) you are using for the input value.
5. Click Next to advance to the Name and Error Handling page.

Defining output parameters and error handling

You create an output name, the data binding name, and how to handle an unsuccessful validation in this page:



You can choose to display an error message if the validation test fails. The error message can appear in a separate dialog box or on the status bar. The message can be a standard message or you may create a custom message.

To create an output name:

- Type an output name in the text field.

To create a fully qualified data binding name

- This field is filled by Visual Cafe as you enter an output name.

To display an error message

- Click either Display a dialog or Post a message to the Log window.

  These settings display only one type of error message. Check both options to display a dialog and post the message to the log window.

To use the default error message

- Click Use default message.

  The default message is “Validation failed for the specified rule.”

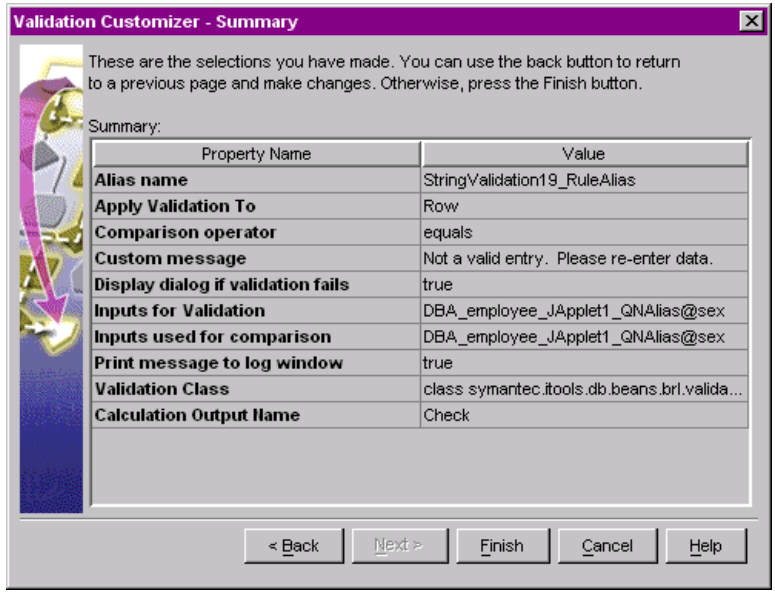
To create a custom error message

1. Click Use custom message.
2. Enter the message text you want to display in the text field.
3. Click Next to advance to the Summary page and review the selections you’ve made with the Add Validation Wizard.

Review your selections

The Summary page of the Validation Customizer displays the choices you’ve made while creating and customizing a `Validation Adapter`.

To change a selection, click Back to go to the appropriate page.



Working with stored procedures

Stored procedures are procedures or functions that you create and store in the database server; they can be used as part of your program.

Adding stored procedures support to your project provides many advantages:

- **Standardization**

  Stored procedures standardize database operations that are used by your applications. By using stored procedures, you don't have to worry about implementation details.

- **Performance**

  Using stored procedures improves the performance of your client/server systems. They are executed in the database and reduce network traffic between clients.

- **Security**

  Stored procedures provide security by limiting user access to database tables. Stored procedures can allow or deny viewing and modifying database information.

Visual Cafe provides a visual environment for working with stored procedures. You can quickly build an application or applet that executes the stored procedure and view the results.

Stored procedure support is added to your project when you use the Add Stored Procedure Wizard. This wizard brings together other JDBC objects, such as, `Stored Procedure Adapter`, which executes a database stored procedure, `Connection Manager`, and `JdbcConnection`, which manages and provides the database connection; `Query Navigator`, which navigates the result set; and `Record Definition`, which caches each data row.

You'll also need to add a `Stored Procedure Trigger` to tell the `Stored Procedure Adapter` to execute the database stored procedure. You can also add a button to your form and use the Interaction Editor to tell the `Stored Procedure Adapter` to execute the database stored procedure. For more information, see "Using a Stored Procedure Trigger" on page 3-84.

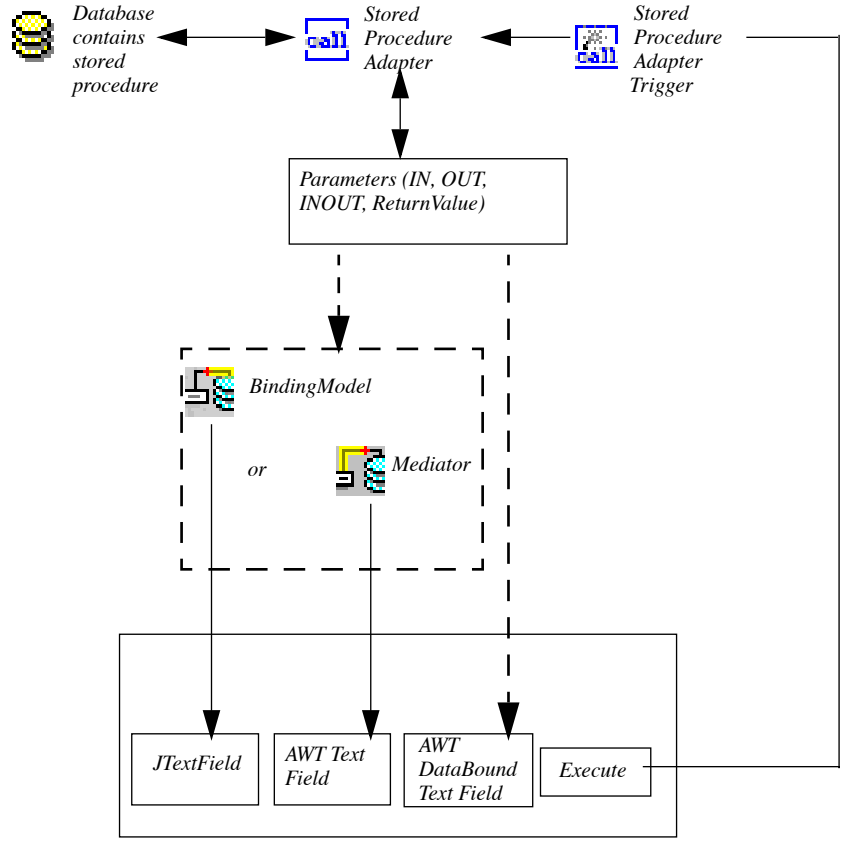
Working with Stored Procedure Adapters

`Stored Procedure Adapters` works similarly to Swing and AWT DataBound components. There are different ways of implementing Stored Procedure Adapters depending on what kind of stored procedure you are working with on the database. The three scenarios presented here are:

- A `Stored Procedure Adapter` that uses parameters only.
- A `Stored Procedure Adapter` that uses result sets only.
- A `Stored Procedure Adapter` that uses parameters and creates result set.

Parameters for Stored Procedure Adapters

The following diagram illustrates a `Stored Procedure Adapter` that uses parameters only:

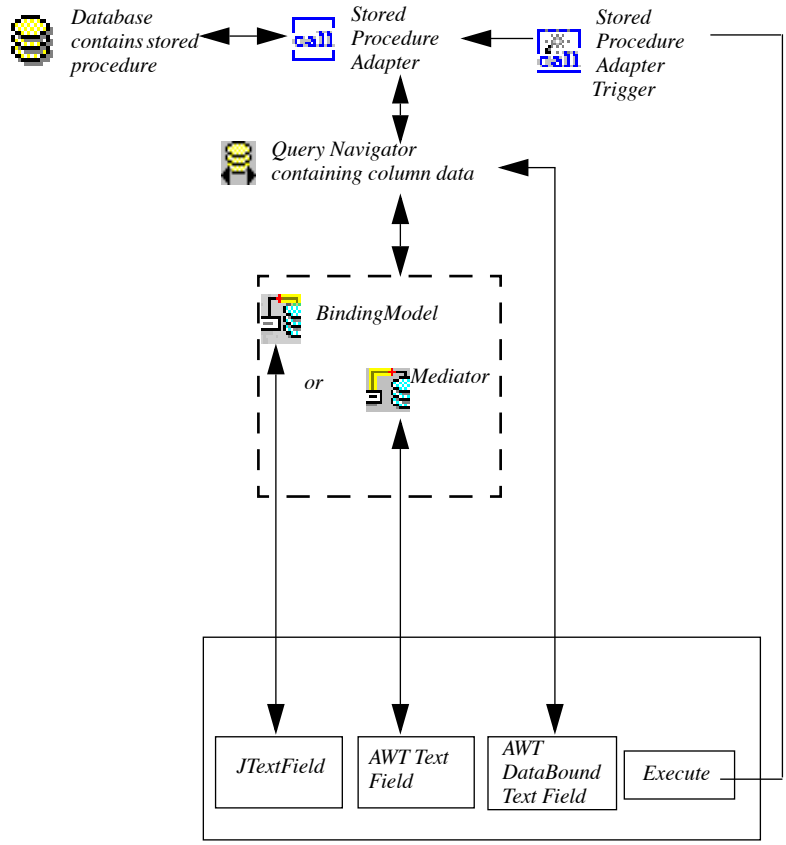


This `Stored Procedure Adapter` uses parameters. Parameter values like `OUT` can become available to other objects that need them. For example, a user could type data in a `JTextField`, as input to a `Stored Procedure Adapter.`

To make the stored procedure work, you'll need to implement a visual object called a `Stored Procedure Trigger`. It calls the `sp.execute()` function that tells the `Stored Procedure Adapter` to trigger execution of the stored procedure in the database.

Result sets from a Stored Procedure Adapter

The following diagram illustrates how result sets are generated by the `Stored Procedure Adapter`.



In this scenario, the `Query Navigator` receives column information from the `Stored Procedure Adapter`. The `Query Navigator` registers the columns onto the `DataBus`, which are then accessed by visual objects as needed.

Stored Procedure Adapter that uses parameters and returns a result set

Sometimes developers need to create database applications that deal with a variety of datasources that pass objects among each other. One common situation is when one object needs data from another object before it can be passed to the client side. In this situation, data changes as a user scrolls through it, and must be displayed in the applet.

In this example, a `Query Navigator` contains data, such as a department ID number, that is passed to an input parameter of a `Stored Procedure Adapter`. This ID number can be passed into a function whose output is a count of the number of employees in a department. Each time Next Record is clicked, the `Query Navigator` goes to the next row. The `Stored Procedure Adapter` automatically gets the new value for the input parameter, in this case, Department ID. The `Stored Procedure Adapter Trigger` calls the `Stored Procedure Adapter.execute()` method to execute the stored procedure with the new input value.

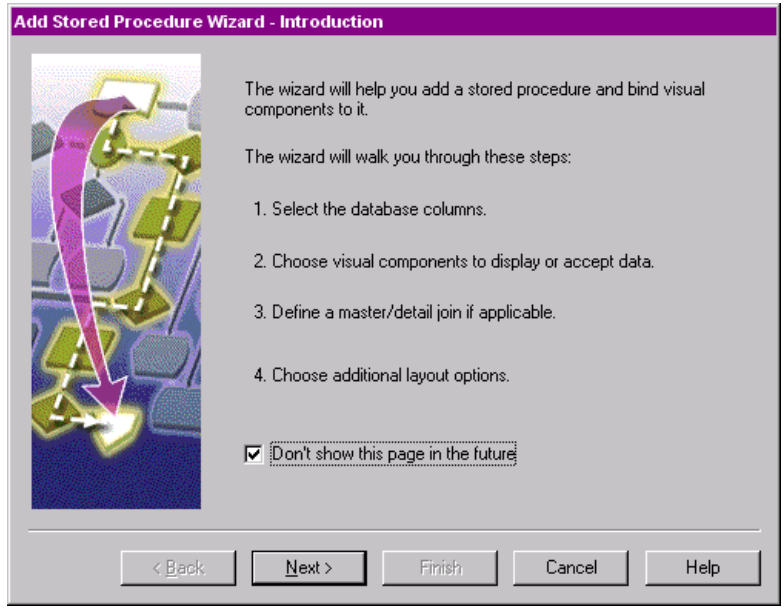
Using the Add Stored Procedure Wizard

When you create a new databound project, you can add database stored procedure support several ways. One way is to add this support while creating a project with the DataBound Project Wizard; using the Add Stored Procedure Wizard, or by using drag-and-drop techniques.

To start the Add Stored Procedure Wizard:

1 From the Database menu, select Add Stored Procedure.

The Introduction page for the Add Stored Procedure Wizard appears:



2 Click Next to select a stored procedure in the database.

To add stored procedure support with the DataBound Project Wizard:

1 See steps 1 - 5 of "Starting the DataBound Project Wizard" on page 3-5.

2 Work through the wizard pages until you reach the Choose a Table page.

3 Select Stored Procedure and then select one from the list.

4 Click Next to test the stored procedure.

For more information about testing the stored procedure, see "Testing the stored procedure" on page 3-74.

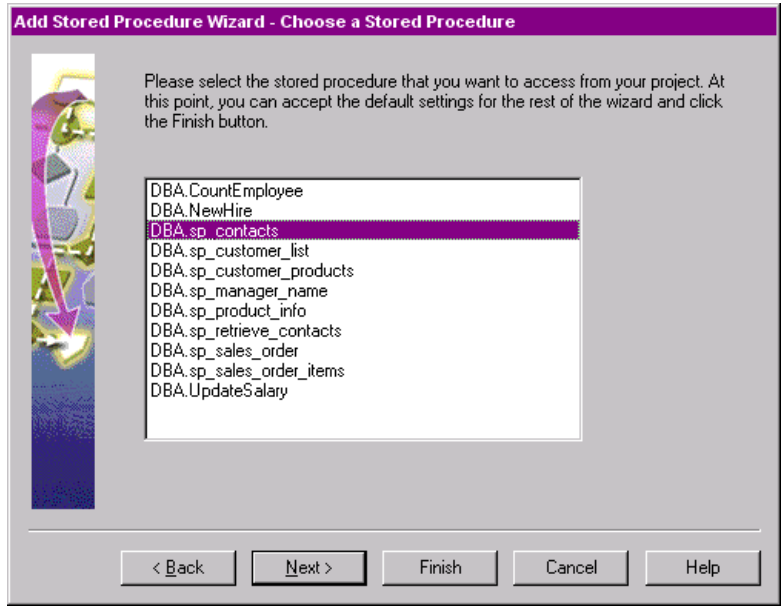
To start the Add Stored Procedure Wizard using drag-and-drop techniques:

1. Drag a `Stored Procedure Adapter` from the Data Source tab of the Component Palette and drop it onto the Form Editor.
2. Right-click on the `Stored Procedure Adapter` and select Customize from the drop-down menu.

   The Add Stored Procedure Wizard appears.
3. Click Next to select a stored procedure in the database.

Selecting the stored procedure

The Choose a Stored Procedure wizard page is where you select the stored procedure to be executed.

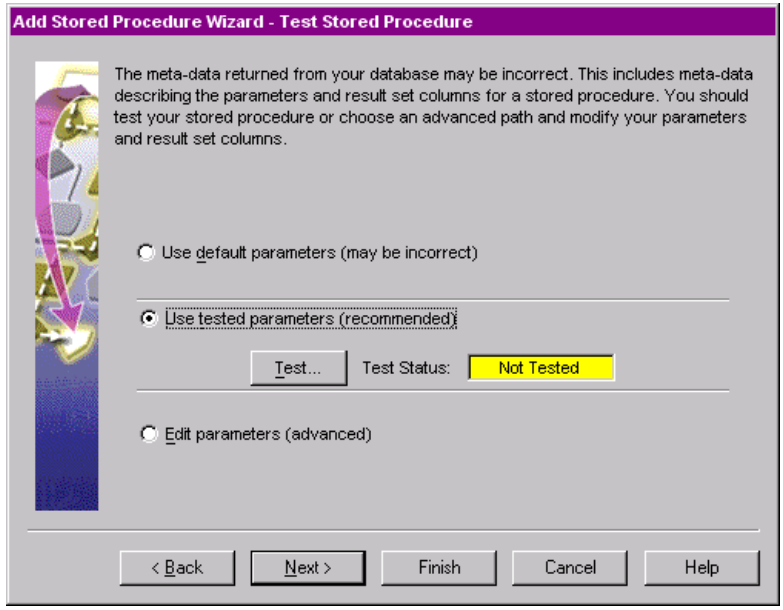


To select the stored procedure:

1. Select a stored procedure from the Stored Procedure window.
2. Click Next to advance to the Test Stored Procedure page.

Testing the stored procedure

You can test the stored procedure you've selected in the Test Stored Procedure page. You can test a selected stored procedure to ensure that the metadata regarding the parameters and result set are correct. Available testing options are: testing metadata with default parameters that the database returns, testing metadata with tested parameters, or editing the parameters yourself.



Caution! Testing the stored procedure may modify the database table by performing inserts, deletes, or changes.

To accept the default parameters and skip testing:

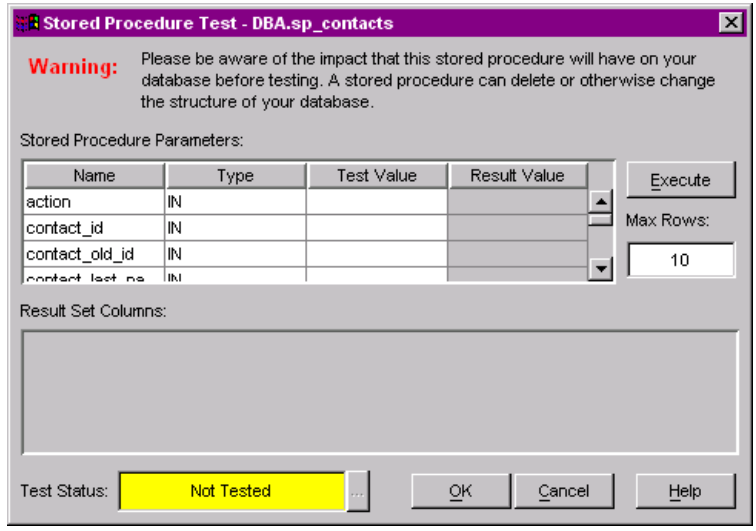
- If you are certain the default parameters found by the JDBC driver are correct, click Next.

  The Choose Parameter Components page appears. For more information, see "Choosing parameter components" on page 3-80.

To test the default parameters:

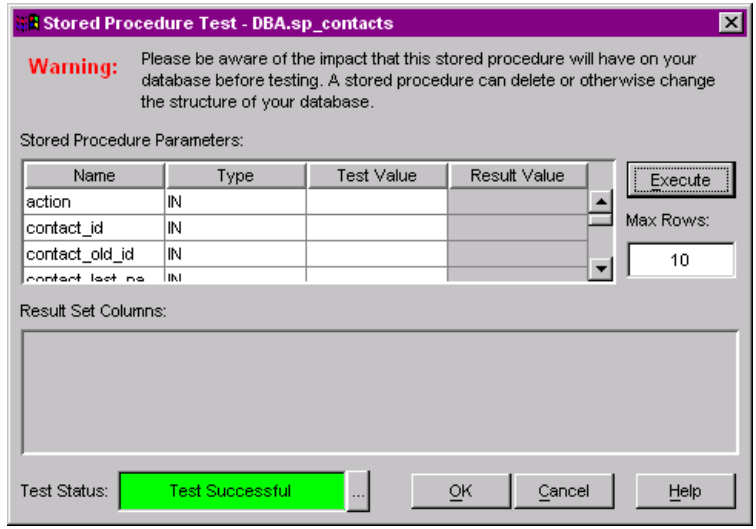
1 Click Test.

The Stored Procedure Test page appears:

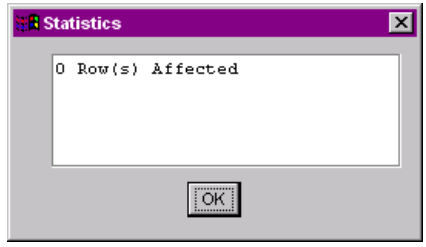


2 Click Execute to test the stored procedure.

If the test is successful, the yellow Test Status field changes to green. If it is unsuccessful, the Test Status field changes to red.



3 If you need to know if any columns were affected by the test, click the ellipsis next to the Test Status text window:



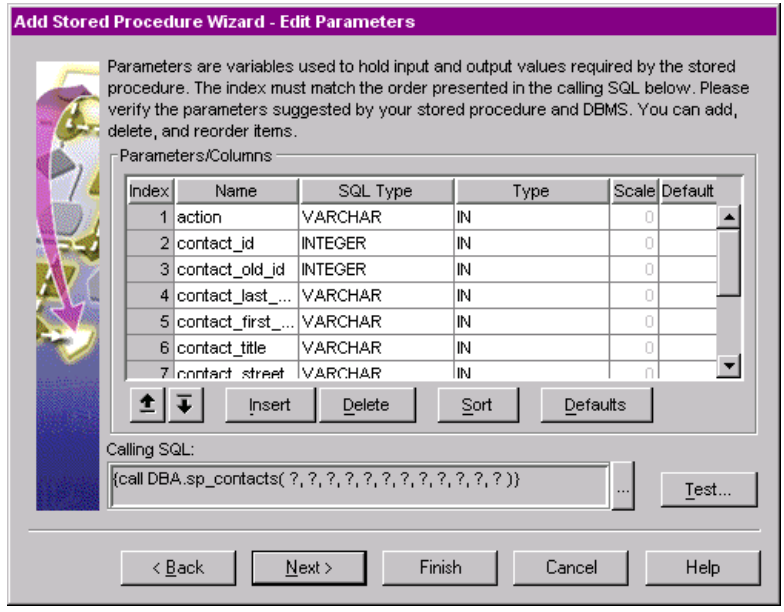
4 Click OK to close the Statistics window.

5 Click OK to return to the Stored Procedure Test page.

6 Click Next to create data bindings for the parameters. For more information about parameter data bindings, see "Creating parameter data bindings" on page 3-80.

Editing parameters

The Edit Parameters page allows you to make changes to the parameters of a `Stored Procedure Adapter`. Parameter information you can change includes the Name, the SQL type, and the parameter type. You may also add or delete parameters, reorder the parameters in the index, sort the parameters, reset the parameters to the default settings and edit the SQL statement that calls the stored procedure. The Edit Parameters page is shown below:



To change a parameter:

1 Click the item you want to change.

2 Enter or select the new information.

To add a parameter:

1. Position the cursor where you want to add a new row. By default, the new row is added at the top of the page if no location is selected.
2. Click Insert.

   A new row appears.
3. Type a new name for the column name.
4. Select a SQL variable type.
5. Select a parameter type.

To delete a parameter:

1. Click on the row you want to delete.
2. Click Delete.

   The row is deleted.

To sort the parameters:

- Click Sort to sort the parameters.
  Parameters are sorted by index number, and result sets move to the end of the index.

To restore defaults:

- Click Defaults.

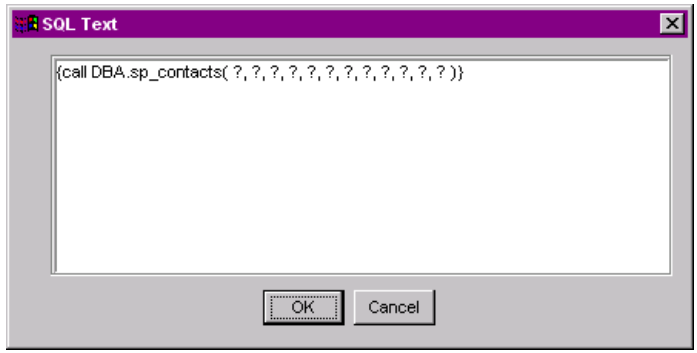
To reorder parameters:

1. Select a row to move.
2. Click Up or Down to move it.

To edit the SQL statement:

1 Click the ellipsis to edit the SQL statement.

The SQL Text Editor displays with the current SQL statement:



2 Edit the SQL statement.

3 Click OK to close the SQL Text Editor.

Note: If you change any of the parameters after you have edited the SQL, the Add Stored Procedure Wizard will override your changes. You should edit the SQL statement *after* you've set all parameters for the metadata.

To test a stored procedure from the Edit Parameters page:

- Click Test to test the new parameters and columns you've selected.

  The Stored Procedure Test page appears. For more information, see "Testing the stored procedure" on page 3-74.

  After the parameters are tested, a status message is displayed in the lower left corner of the dialog box. If you receive a `test failed` error message, it is recommended that you edit the parameters using the Edit Parameters page. For more information, see "Editing parameters" on page 3-77.

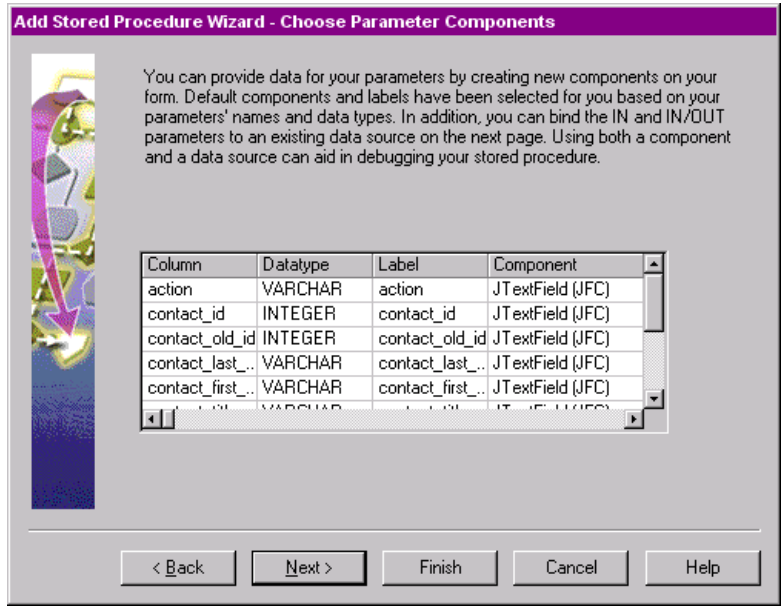
4 Click Next to select visual components for your parameters.

Choosing parameter components

In the Choose Parameter Components page, you select visual components that will display the data for your parameters on your form.

To select the visual components for parameters:

1. Click the Component field and select a component for a column.
2. Click Next to create the data bindings between the components and datasources.

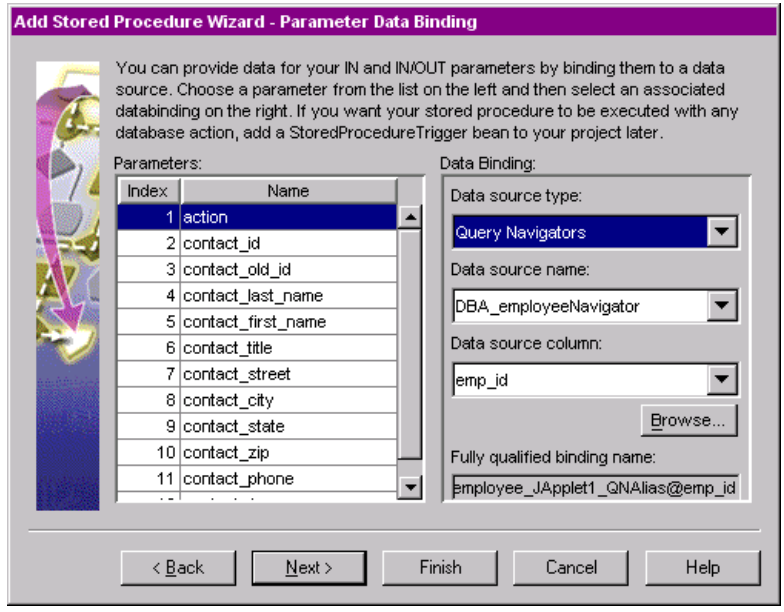


Creating parameter data bindings

You use the Parameter Data Binding page to link input data to a parameter from a datasource. The available datasources include parameters from the current stored procedure, a different stored procedure, data from a `Calculation Adapter` or `Validation Adapter`, or data from a database table using a `Query Navigator`. You can skip this page if you want to manually enter data from the applet or application at run time.

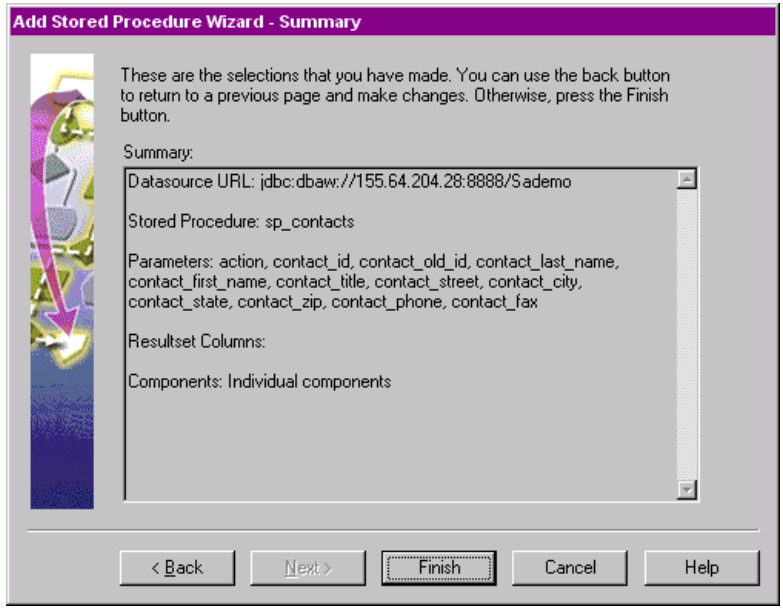
To specify data to be bound to a parameter:

1. Select the parameter from the left side of the Parameter Data Binding page.
2. Select the data binding information from the appropriate list boxes on the right side of the Choose Parameter Components. If you want to add a new datasource, click Browse and add it from the Data Binding Editor or the dcNAVIGATOR. For information, see "Using the Data Connection Navigator" on page 3-41.
3. Click Next to create the data binding.



Reviewing your selections

You can review the selections you have made on the Summary page. If you are satisfied with your selections, click Finish to customize the `Stored Procedure Adapter`. This information will be reflected in the Property List for this project. For more information about modifying these properties, see "Modifying properties" on page 3-21.



Creating custom Stored Procedure Adapters

The DataBound Project Wizard and the the Add Stored Procedure Wizard simplify much of the work in adding stored procedure support to your project.

If you wish, you can manually implement a customized `Stored Procedure Adapter` using drag-and-drop techniques.

Dragging and dropping the required components

You'll need to establish a datasource before you add the `Stored Procedure Adapter`. You must also manually set the properties of the following components in the Property List.

To add components for a Stored Procedure Adapter:

1 From the Data Source tab of the Component Palette, or from the Component Library, select a `Connection Manager` component and drag it into the Project Window.

2 Drag a `JdbcConnection` component into the `Connection Manager` component in the Project Window and set the following properties:

- `URL` (selecting a datasource)
- `Username`
- `Password`

3 From the Data Source tab, drag a `Stored Procedure Adapter` and drop it onto the Form Designer.

4 Click on the `Stored Procedure Adapter` icon.

5 Set the properties of the `Stored Procedure Adapter` in the Property List.

Working with result sets

If your datasource returns result sets, you'll need to add components to manage the result set. In addition, you must set the properties for them in the Property List.

To add support for result sets:

1 Drag and drop a `Query Navigator` component into the Form Designer.

2 Drag and drop a `SQL Record Definition` component into the Form Designer.

3 Set the following properties for the `Query Navigator`:

- `Alias Name`
- `Record Definition`
- `SQL Adapter`

The `SQL Adapter` property must point to the `Stored Procedure Adapter`.

4 Drag buttons and drop them onto the form for executing the stored procedure. Use the Interaction Editor to set up all the interactions.

5 Add visual components so your program can display data and accept input. Use a `Binder Model` if you are using Swing components, or a `Mediator` if you are using non-databound AWT components.

Using a Stored Procedure Trigger

A `Stored Procedure Trigger` is a component that communicates with a `Query Navigator` and a `Stored Procedure Adapter`. The `Stored Procedure Trigger` listens to the `Query Navigator` for any changes in data objects – such as input from a user. The `Stored Procedure Trigger` communicates to the `Stored Procedure Adapter` to execute the stored procedure from the database when a condition is satisfied.

To implement a `Stored Procedure Trigger`, you must have an instance of a `Query Navigator` in your project. When you define a `Stored Procedure Adapter` in your project, a `Query Navigator` is defined automatically. For more information, see "Selecting tables or stored procedures" on page 3-13.

To insert a Stored Procedure Trigger:

1 Add a `Stored Procedure Adapter` to your project and customize it for the stored procedure you want to execute.

2 From the Data Source tab of the Component Palette, drag a `Stored Procedure Trigger` component and drop it onto the open form that contains the `Stored Procedure Adapter` it will trigger.

3 In the Property List, Select the `Query Navigator` that is bound to the `Stored Procedure Adapter` you want to trigger.

4 In the Property List select the appropriate `Query Navigator` and `Stored Procedure Adapter` properties.

Handling exceptions

The `Query Navigator` component registers itself as a listener for all types of exceptions encountered within all classes handled by `Query Navigator`. This code is in the constructor of `Query Navigator`:

```
public Query Navigator(){

    ...

    addExceptionEventListener(this);
}
```

A `Query Navigator` handles the encountered exceptions by implementing:

```
public void handleExceptionEvent(ExceptionEvent e){
    System.out.println(e.getException());
}
```

When this code is implemented, the default behavior of the application is to flush all the errors to `System.out`.

The application never stops; all exceptions are handled. However, if an `APPLICATION_EXCEPTION` occurs, that means something that was not expected by the code has happened and the application is no longer under your control. You probably should restart the application.

Changing where your Java program displays errors

To change where errors are displayed, you need to register a listener for `Query Navigator` in the Java program code. For example, if you want to show errors in a pop-up window, add the following source code:

```
public void init(){

    ...

    Query Navigator1.addExceptionEventListener(window);
}
```

The `window` can be an instance of a particular Window class that implements the interface `ExceptionEventListener`.

To implement `ExceptionEventListener`, the `Window` class has to provide an implementation of the `HandleExceptionEvent` method. This method decides where and how to display the error message:

```
public void handleExceptionEvent(ExceptionEvent e){
   window.draw(e.getExeption(),5,5);
}
```

The `ExceptionEvent` class defines several error types, including `APPLICATION_EXCEPTION`, `DB_EXCEPTION`, and so on, so that you can tune your event handler according to the type of the exception that's encountered.

Handling last row, first row, and no rows exceptions

Sometimes an exception occurs when the user tries to perform an invalid action, such as clicking a Next button where there are no further rows.

Here are some examples:

- The end of the `RecordSet` was reached and Next is clicked.

  The `Query Navigator` triggers an `ExceptionEvent` of the `EXCEPTION_LASTROW` type.

- The beginning of the `RecordSet` was reached and Previous is clicked.

  The `Query Navigator` fires an `ExceptionEvent` of the `EXCEPTION_FIRSTROW` type.

- The `RecordSet` has no rows.

  The `Query Navigator` triggers an `ExceptionEvent` of the `EXCEPTION_NOROWS` type.

By default, the `Query Navigator` component handles these events itself by writing the appropriate message to `System.out`. If you want to handle these events differently, you must register a listener for the `Query Navigator`.

Here's some sample source code:

```
public init(){

    ...

    Query Navigator1.addExceptionEventListener(this);
}

public handleExceptionEvent(ExceptionEvent e){
    if (e.getEventType==ExceptionEvent.EXCEPTION_NOROWS) {
        //disable the FIRST button
        btnFirst.setEnabled(false);
        //Move the user to a valid row, in case they want to enter
 data
        Query Navigator1.new();
    }
    else if(e.getEventType==ExceptionEvent.EXCEPTION_FIRSTROW) {
        //disable the PREVIOUS button
        btnPrevious.setEnabled(false);
    }
    else if(e.getEventType==ExceptionEvent.EXCEPTION_LASTROW) {
        //disable the NEXT button
        btnNext.setEnabled(false);
    }
```

C H A P T E R 4

Data Binding

Data binding is connecting your visual components with data sources for displaying and manipulating data. There are several types of data binding that you will need to do, depending on the types of components you're using.

Information in this chapter includes:

- Basic database concepts
- An overview of the data models inVisual Cafe database components
- An overview of the datasource components
- Data Binding and displaying data in your Visual Cafe project

Basic database concepts

This section describes some basic database concepts you should be familiar with before creating and working with databound projects. If you are already familiar with Swing and data binding, you can read about some of the databound components you will be working with. For more information, see "Datasource components" on page 4-13.

What is SQL?

The **Structured Query Language (SQL)** is a specialized language for organizing, managing, and retrieving data stored in a relational database. It has emerged as the standard tool for managing data on personal

computers, minicomputers, and mainframes. All JDBC drivers (see "What is JDBC?" on page 4-2 for more information) support a particular subset of the SQL standard.

What is a relational database?

A **relational database** is a database that is built on the Relational Data Model (RDM). The word *relational* is derived from the concept of s*et relation*, which is part of Set Theory, which in turn is part of RDM. In a relational database, data is stored as relations and represented as tables to the user. These tables are broken down into rows and columns.

You access the information in a database using its *engine*, or server. The engine can be located on the same computer as the application program accessing it, or can be established elsewhere on a network.

The users of a relational database are typically divided into two categories: administrators and end users. The database administrator has the ability to manipulate the structure of the database, adding or removing a table or column, for example. The database user is generally constrained to manipulating or viewing the data stored in the database.

The following list includes some of the operations you can perform as the user of a relational database system:

- Viewing records
- Adding or removing records
- Inserting new data into existing records

The databound Java programs that you create with Visual Cafe are typically oriented toward the database user, not the administrator.

What is JDBC?

Visual Cafe generates Java DataBase Connectivity (JDBC) code to connect you to a database. The Database Edition has added some of its own classes and interfaces to the JDBC standard set of classes and interfaces, which were created by Sun Microsystems to extend the Java language. These extensions allow your application or applet to "talk" to a database and "share" information with it.

Specifically, JDBC is a Java API for executing SQL statements. By using it you can communicate with virtually any relational database. JDBC carries forward the Java promise of “write once, run anywhere” into the realm of relational databases.

What can JDBC do?

By creating an applet or application using Visual Cafe to generate JDBC code you gain the ability to:

- connect to a database
- send SQL statements to query your database
- generate query results
- perform inserts, updates, and deletions
- execute stored procedures

A databound applet could be used to implement a corporate phone list or an e-mail form that allows you to receive feedback from a Web page. A databound application might implement a stock reporter that gathers information from a database on the Web and stores it in a database on your local machine.

Understanding data models in Visual Cafe

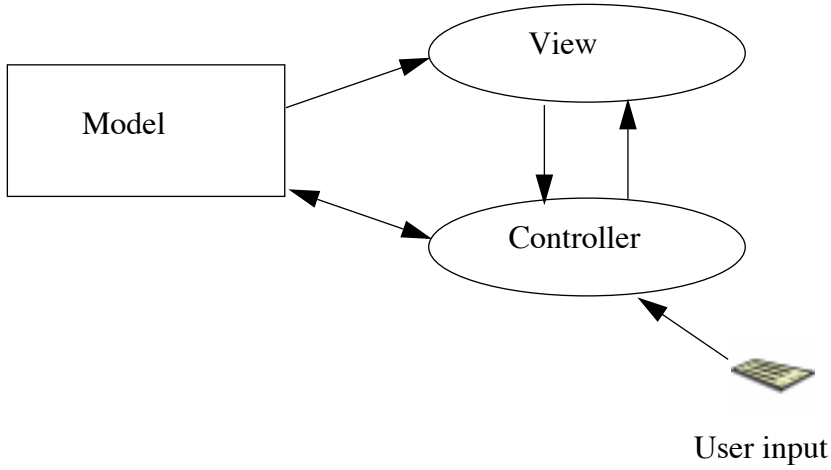
If you’re new to database design, you might want to read about the various models that Visual Cafe uses to help you develop your projects.

This section describes:

- Data models in Visual Cafe
- datasource components
- General descriptions of Swing and AWT DataBound components
- Binding datasources with visual objects, such as AWT and Swing components

Data models in Visual Cafe

Visual Cafe uses a **Model-View-Controller (MVC)** design to pass data to your visual objects. The following diagram provides an overview of MVC architecture:



MVC is an architecture for developing interactive applications. It requires applications to implement a modular design which allows them to be scalable and maintainable.

Model

The `model` component holds the information (data) that the application is gathering, displaying, and manipulating.

View

The `view` is the visual display component of the `model`. There can be multiple `views` at any time. As data changes in the `model`, the `view` is notified so that it changes to reflect the contents or current state of the `model`.

Controller

The `controller` receives input events from the user. Once the controller has received input, it communicates with the `view` object to determine which objects are changing. Then, the `controller` calls the appropriate methods in the `model` and the `model` notifies the `view` to update.

Working with Record Definition components

The `Record Definition` component defines and accesses a row of data in a database table. The `Record Definition` is not part of a specific form or applet; rather it is part of the project. Therefore, it can be used by multiple forms. The `Record Definition` requires objects to be implemented for each of the following properties:

Note: Do not change the properties of a `Record Definition` component at runtime.

Working with Jdbc Connection components

The `Jdbc Connection` component represents a connection to a specific datasource. It has properties for establishing connection parameters in accordance with the JDBC standard.

Dragging a datasource item from dbNAVIGATOR to a project creates `Jdbc Connection` objects. You have one `Jdbc Connection` object per datasource. All of the `Jdbc Connections` object in the project are managed by a single `Connection Manager` object.

The ConnectFailedListener property defines the class of the object that handles failed attempts to connect with the server.

- The JDBCDriver property defines the class of the JDBC driver used.
- ReadOnly property determines whether the database may be modified through this connection. If true, the database may not be modified.
- URL property defines the database server's URL.
- UserName property is the name used to try to connect with the server.
- UserPassword property is the password to try to connect with the server.

Note: Do not change Jdbc Connection component properties at runtime.

Creating custom database logons

Visual Cafe Database Edition provides a default logon dialog box. You can specify your own dialog box, if needed.

The Jdbc Connection component specifies a logon dialog box. To use a different logon dialog box, you can create your own logon dialog class that uses the ConnectFailedEvent, ConnectFailedListener, and connectFailed method and specify it for the Jdbc Connection component.

First, you can use the Insert Class Wizard, specifying the name of your logon dialog class and adding the interface symantec.itools.db.beans.jdbc.ConnectFailedListener (See the Visual Cafe User's Guide for more information). Then you can modify the code as needed. Here is sample code that results from using the Insert Class Wizard:

```
import java.lang.*;
import symantec.itools.db.beans.jdbc.*;
public class MyLogon extends java.lang.Object implements
 symantec.itools.db.beans.jdbc.ConnectFailedListener {
public void connectFailed(ConnectFailedEvent event) throws
 Exception {
//You must add your code here.
}
}
```

In the Property List, modify the Jdbc Connection component's Connect FailedListener field so it is the name of your logon class (for example, MyLogon).

In your logon class, once the user has entered a password and user name, you should call:

```
Jdbc Connection connection = (Jdbc
 Connection)event.getSource();
connection.setUserName(The_New_User_Name);
connection.setPassword(The_New_Password);
connection.connect();
```

Working with Connection Manager components

The **Connection Manager** component manages and logically contains the `Jdbc Connection` components for a project. It is a top-level non-visual bean, meaning that it is not contained within any form.

Dragging a datasource (e.g. a table) from the dbNAVIGATOR window to a project adds a `Connection Manager` object and a `Jdbc Connection` object to the project. There is one `Connection Manager` object per project. You can drag a `Connection Manager` object to the Component Library.

You have one `Connection Manager` object per project, and one `Jdbc Connection` object per datasource.

Working with Mediator components

The `mediator` component lets you make a standard component (such as AWT components) behave like a databound component. It serves as a bridge between the component and the `Query Navigator` component. It is found on the AWT DataBound tab of the Component Palette.

When you use an AWT DataBound component, the `mediator` is already used as part of these components. You may be using them even if they are not visible in your project.

At design time, you can add a `mediator` component to a form in a project and set its properties for a particular component on the form. If you are creating a JavaBeans component or have access to the component's Java code, you can add a `mediator` to the component code.

You could have two mediators for one component. For example, if you have a list of states in the United States, one mediator could be the column and the other could be the lookup. The column mediator is the data binding — it is the one used by the database wizards and the Database Environment Options.

Using a non-databound component with a mediator

At design time, you might want to bind a non-databound component if, for example, you have a component from another vendor and you want to use it with a database. Alternatively, you can add the mediator to the component Java code.

To work with a mediator and a non-databound component:

1. Start with a project that has the databound components that manage the connection with a datasource. For example, you can use DataBound Project Wizard or Add Table Wizard, or drag components from dbNAVIGATOR.

Tip: You could also add these databound components after adding the mediator, then adjust the Data Binding property of the mediator accordingly.

---

2. Add a non-databound component to a form.
3. Add a mediator to the component by adding this line of code:

```
private Mediator    m_Mediator = new Mediator();
```

4. In the constructor of your component, set the properties of the mediator, as shown in the sample:

```
m_Mediator.setOutput(this);
m_Mediator.setSetMethods(m_SetMethods);
m_Mediator.setGetMethods(m_GetMethods);
```

Note: You must add the mediator after you add the component.

---

5 Add the methods for setting the `Data Binding` property using code similar to the following example:

```
public void setData Binding(String binding) {
    m_Mediator.setData Binding(binding);
}

public String getData Binding() {
    return m_Mediator.getData Binding();
}
```

Tip: Use the Class Browser as an easy way to look up the methods in a component.

These methods should follow the standard coding guidelines for properties. You should also expose the `Data Binding` property in the `BeanInfo` so your users can set it.

Property	Description
`Output`	Name of the non-databound component.
`SetMethods`	String array property specifying the methods the mediator should use to provide the non-databound component with data (depends on methods used in component). Each method should be of the format: `method-name(value [, row] [, col] )`; for example: `{"setLabel(Value)"}`. The method or methods are enclosed by `{"` and `"}`, and delimited with a comma.
`GetMethods`	String array property for obtaining data from the component (depends on component methods). Each method should be of the format: `method-name( [row] [, col] )`. An example is `{"getLabel()"}`. The method or methods are enclosed by `{"` and `"}`, and delimited with a comma.
`Data Binding`	String property made of an alias name for the `Query Navigator` followed by the column name, number, or both (`ALL` is a valid value). For example, `Query Navigator1@customer@last_name%5`, where `5` is the number of rows to display.

Creating a databound component with a mediator

To create a databound component, the `mediator` is defined within the component. Alternatively, at design time you can bind a non-databound component.

Here is a code sample that shows the mediator within a component:

```
import java.awt.*;
import symantec.itools.db.beans.binding.Mediator;

public final class DataAwareButton extends java.awt.Button {
    private Mediator    m_Mediator = new Mediator();
    private String[]    m_GetMethods={"getLabel()"};
    private String[]    m_SetMethods={"setLabel(Value)"};

    public DataAwareButton() {
        this("");
    }

    public DataAwareButton(String label) {
        super(label);
        m_Mediator.setOutput(this);
        m_Mediator.setSetMethods(m_SetMethods);
        m_Mediator.setGetMethods(m_GetMethods);
    }

    public void setData Binding(String binding) {
        m_Mediator.setData Binding(binding);
    }

    public string getData Binding() {
        return m_Mediator.getData Binding();
    }
}
```

Working with Validation Adapter components

The `Validation Adapter` is a JavaBean that helps you add client-side validatation to your data. If data validation fails for a corresponding criteria, the user is warned by displaying a dialog box that indicates that the validation failed. You can use `Validation Adapters` for any number of purposes. For more information on working with `Validation Adapters`, see "Calculation and validation rules" on page 3-52.

Working with Calculation Adapter components

The `Calculation Adapter` is a JavaBean that allows you to add mathematical functionality to your project. You can use predefined calculations, or import your own custom Java class. For more information on working with Calculation Adapters, see "Working with stored procedures" on page 3-67.

Working with Stored Procedure Adapter components

Stored procedures are predefined and compiled functions that are stored in a database. You use a `Stored Procedure Adapter` to add them to your databound project. For more information on working with Stored Procedure Adapters, see "Working with stored procedures" on page 3-67.

Data binding in Visual Cafe

A databound project has components that are bound to database columns or data sources. The process of connecting these database elements with Visual Cafe components is called data binding. Visual Cafe binds components to database elements through the JDBC API (a standard Java interface). The JDBC API is a set of classes developed by JavaSoft as a standard SQL-level database protocol. It enables the Java developer to access relational databases in purely Java terms. It is based on the X/Open SQL Call-Level Interface.

The underlying mechanism that facilitates data binding in Visual Cafe is called the **DataBus**. Visual Cafe has packaged the interfacing between the

DataBus and the components that are not already databound as a component called a mediator, which mediates the communication between the database and the component. You can also bind other components, such as those defined in the JFC, to database elements, making them databound. To do this, you must connect a JFC component to a `BindingModel` explicitly.

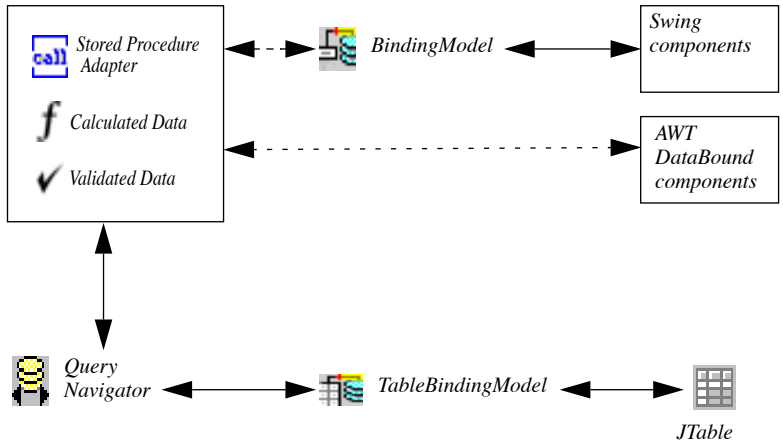
The AWT DataBound tab of the component palette offers a number of databound components that you can use with the JDBC API, such as `TextField`, `TextArea`, `Grid`, `ComboBox`, `CheckBox`, `RadioButton`, and `ImageViewer`.

Datasource components

Datasource components include adapters as well as non-visual components. A non-visual component is a component that doesn't appear to the user at run-time, though it may appear on a form at design time. There are several non-visual components that you'll interact with while developing your databound project.

Adapters are Beans that allow you to add functionality to your project from sources outside the project, such as calculation and validation rules. You can also use adapters to extend database functionality, with stored procedures, for instance. to your project.

The following illustration shows how databound components (Swing DataBound and AWT DataBound) work with datasource components:



Working with Query Navigator

The `Query Navigator` component manages the set of records which you retrieved from a database. Dragging a database table from dbNAVIGATOR to a form in a project creates a top-level `Record Definition` object and a `Query Navigator` object. The `Query Navigator` object is contained by the form. The `Query Navigator` component supports Query-By-Example, including sorting and the SQL `WHERE` clause.

A project can contain more than one `Query Navigator`. These views can have a joined relationship, or they can be independent. In the case of a join, the first `Query Navigator` in a project is used as a master view.

When you add a second `Query Navigator` component, you create a detail view, thereby forming a master-detail relationship. The join relationship between the two `Query Navigator` components is created by specifying `MasterAliasName` and other associated `Join` properties. For more information about working with master-detail relationships, see "Working with Master-Detail definitions" on page 3-28.

Query Navigator properties

The following properties are part of a `Query Navigator` component:

- `AliasName`
  Gives this object a name so it may be referenced by other objects.
- `Filter`
  Constrains the data displayed in the component.
- `Join`
  Used to specify a master-detail relationship with another Query Navigator.
- `SortOrder`
  Specifies the order in which to sort the displayed data.

Note: Do not change Query Navigator component properties at runtime.

Managing Query Navigator components

The `Query Navigator` component is contained by a form and provides data binding for the form. The `Query Navigator` component obtains and maintains data.

When using `Query Navigator` components, you need to add code to:

- Close the `Query Navigator` when the form that contains it closes.
  A `Query Navigator` is referenced by internal components, so it is not closed automatically when the form that contains it is closed. It is good programming practice to close a `Query Navigator` when its context is destroyed; this will clean up the `Query Navigator` and let garbage collection occur.
- Close the form containing a `Query Navigator` when the `Query Navigator` closes.

Using unique Alias Name properties

You can use a `Query Navigator` across multiple forms. For example, in a cross-form join each `Query Navigator` provides a unique `AliasName` property by which it can be located.

Two `Query Navigators` can't use the same `AliasName`. If this is attempted, the newly added `Query Navigator` takes the place of the first `Query`

Navigator, and garbage collection is performed. This means that if two copies of a form are created without offering a unique "per instance" AliasName for each form's Query Navigator, then the original form stops functioning.

Closing a detail form when the master form closes

A Query Navigator can depend on a parent Query Navigator, for example, when the Query Navigator is the detail in a master-detail relationship. In your code, when you receive a property change event (java.beans.PropertyChangeEvent), you can close the Query Navigator form when its master Query Navigator is closed.

For example, if there is a master form and a detail form, and you can code the master form to call close()on the master Query Navigator when the form is closed. The following actions occur when the form is closed:

1. The master form closes.
2. The master form closes the master Query Navigator.
3. The master Query Navigator notifies the detail Query Navigator.
4. The detail Query Navigator closes itself because it is no longer valid without its parent.
5. After being notified that its Query Navigator closed, the detail form closes.

Note: When using this scenario, you *must* program steps 2 and 5 into your code.

Updating a cached detail set

In a master-detail relationship, detail records display if they comply with the relationship. In practice, this methodology can be too strict. For example, when you change or add a detail record such that it does not match the current detail set, the record appears in the cached detail set until you save it. Saving the record updates it in the database, after which it is removed from the cached result set. However, this is only the case if the join operator is equal (=) or not equal (!=). It is not removed for all other operators, such as <.

To ensure that the cache is always accurate, issue a Query Navigator restart() call after a save operation. Or, use the Query Navigator

`saveAll()` method, which saves all of the changes in the cache and restarts the result set.

Using an Auto Start property of false

Setting the `AutoStart` property to `true` causes a `Query Navigator` to issue its query as soon as it has been instantiated. You can turn off this behavior by setting the `AutoStart` property to `false`, which can be useful in certain cases, as described in these examples:

- You are using the `Query Navigator` to insert records into a database table, but a query is never required.
- You use the `Query Navigator` to display a detail result set, but you want to disable the detail set until you issue a `restart()` call. For example, you might want to hide the existence of a detail result set for performance reasons or to conserve screen space. When the detail result set is needed, you can issue a `restart()` call on the `Query Navigator`, at which point the detail result set will begin issuing queries automatically. The detail `Query Navigator` does not react to scrolling of the master `Query Navigator` until a restart message is sent.

Closing a form when its Query Navigator closes

The `Query Navigator` component provides data binding to a form. You can add code that closes a form when a `Query Navigator` contained by it closes. Following is a quick way to do this by using the Interaction Editor.

To close a form through the Query Navigator:

1. Select the `Query Navigator` component in the Objects view of the Project Window.
2. Choose Object, then Add Interaction.

   The Interaction Editor appears.
3. In the Interaction Editor, verify that the trigger `Query Navigator` component appears in the Start an interaction field title. If it does not appear, you need to start over.
4. In the Start an interaction field, select `propertyChange` as the triggering event that activates the interaction.
5. In the Select the item you want to interact with field, choose the form containing the `Query Navigator`.

6 In the Choose what you want to happen field, select an action (such as `Disable`). Choose an action that takes no parameters so that Next is disabled.

7 Click Finish.

In the Java source file, code is generated for the call handler, listener registration, and adapter/listener class.

In the source code, Visual Cafe performs several edits:

1 Visual Cafe generates an adapter or listener implementation for the event. If one was already generated, it is used with the new interaction.

2 In the adapter/listener class, Visual Cafe generates a check for the object requesting the handling of the event and a call to the event handler.

3 Visual Cafe instantiates this adapter/listener class and generates the registration (specifically, object.addtypeListener) after the `REGISTER_LISTENERS` tag. If the adapter/listener class has already been instantiated, it is used.

4 An event handler is generated. If the event handler already exists, it is used.

5 The interaction specified in the wizard is generated in the event handler.

6 Overwrite the event-handler code with this line:

```
dispose();
```

For example:

```
void
 DBADetailNavigator_propertyChange(java.beans.PropertyChange
 Event event)
{
// to do: code goes here.
dispose();
}
```

About JFC

The Java Foundation Classes Library is a collection of graphical user interface APIs. These APIs were designed to provide a more polished and mature look and feel than what the AWT classes could offer. These APIs include: `AWT`, `Java 2D`, `Accessibility`, `Drag and Drop`, and `Swing`.

JFC data formatting

JFC components are designed very differently from their AWT predecessors. Because JFC is patterned on the Model-View-Controller architecture (see "Data models in Visual Cafe" on page 4-4), JFC components separate the front-end display of data from the data storage. In JFC components, the GUI component is reduced to its minimal and essential content, the view. Swing components now rely on the developer to provide methods and data structures govern and store data. In this structure, the formatting and data operations are executed in the data storage or model component. The physical components simply display results or receive input from the user.

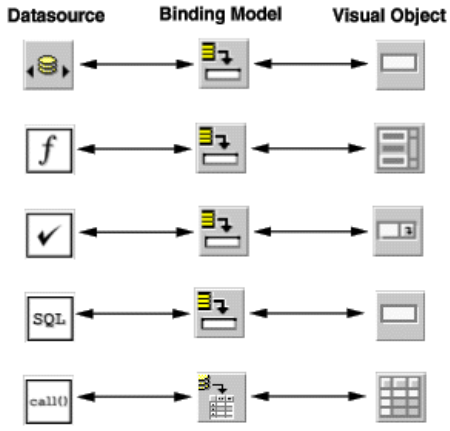
Because of this architecture, JFC display components are supposed to be as ignorant as possible of the content of the data they display. The text components come fully implemented with string-manipulation methods that allow you to have standard editing functionality, such as copy, cut, paste, select, and so on. However, any custom operations must be executed by the `DataModel` class. JavaSoft furnishes a default implementation for some of the interfaces; but developers must implement the necessary logic for operations, such as date/time formatting, integer, float formatting, currency operations, validation and derivation.

Visual Cafe makes implementing JFC classes easier when you use the `BindingModel` that is located on the Data Source tab of the Component Palette. It creates the data binding between the DataBus and the JFC component requesting data from it. For more information, see Appendix A, "Understanding the DataBus".

Since the `BindingModel` class implements most of the common JFC model interfaces, it can be passed directly to most of the components without restriction.

Binding Swing components

You can also bind other components, such as Swing components, to database elements, making them databound. Visual Cafe provides a component called `BindingModel` that you use to establish an interface between the data sources and components that aren't already databound. Like `Mediator`, `BindingModel` mediates the communication between the database and the Swing component. The following diagram is an overview of how Swing components are bound to data sources:



In the left column, there are five possible data sources. The `BindingModel` in the center of the diagram binds the data to the specified components. You specify these components with the Data Binding Editor. For more information, see "Table data binding" on page 4-26.

Visual Cafe makes binding Swing components easier. The procedure is very similar to binding an AWT component using a `mediator`. The general procedures are described below:

- Drop a Swing component and a `BindingModel` on the form. The order in which the Beans are dropped does not matter.
- Drop a Data Source component and configure its properties.
- Set the `BindingModel` properties. This Bean has three important properties: `Data Binding`, `LookupBinding`, and `Component`.
- The `Data Binding` property represents the main link for the component. This link contains only a single row of data and is the main link of a `TextField`, the selected item in a `TextArea`, or the current item in a `ComboBox`.
- A `LookupBinding` property represents the secondary link of the component. It populates a `List`, `ComboBox`, or other component containing static data. Its size is left up to the user, even if in the most common scenarios it contains all the rows present in the datasource.
- The `Component` property identifies the Swing component for data binding.

For information on how to set `BindingModel` properties, see "Working with SQL" on page 5-1.

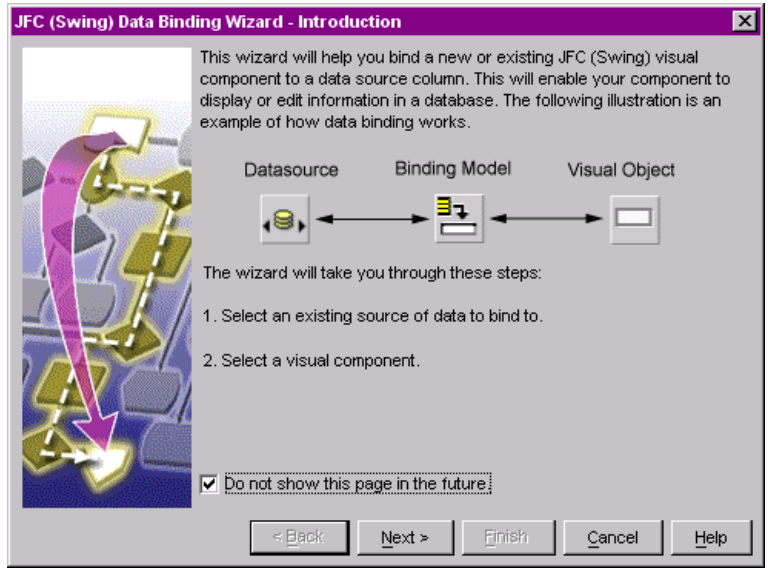
Using the Add/Data Bind Component Wizard

The Add/Data Bind Component Wizard is designed to help you bind a Swing component to a datasource. You can use it to add data to a single component from an existing datasource. Data sources you might typically bind to a component include `Query Navigators`, `Stored Procedure Adapters`, and `Calculation Adapters`.

To start the Add/Databind Component Wizard:

1. From the Swing tab of the Component Palette, drag a component and drop it onto the form.
2. From the Database menu, select Add/Databind Component

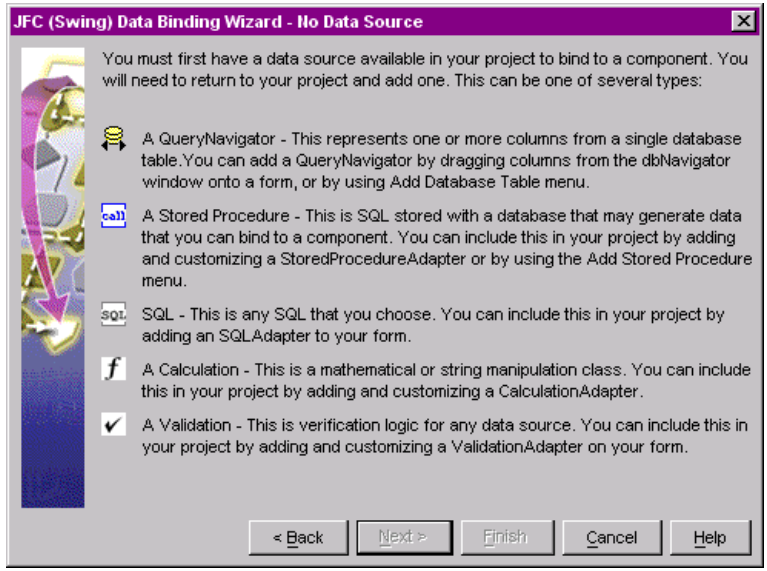
   The Add/Data Bind Component Wizard Introduction page appears:



3. Click Next to select a datasource.

Selecting a datasource

You must select an existing datasource with which to bind your Swing component. If you haven't defined a datasource for this project, the No datasource page appears:



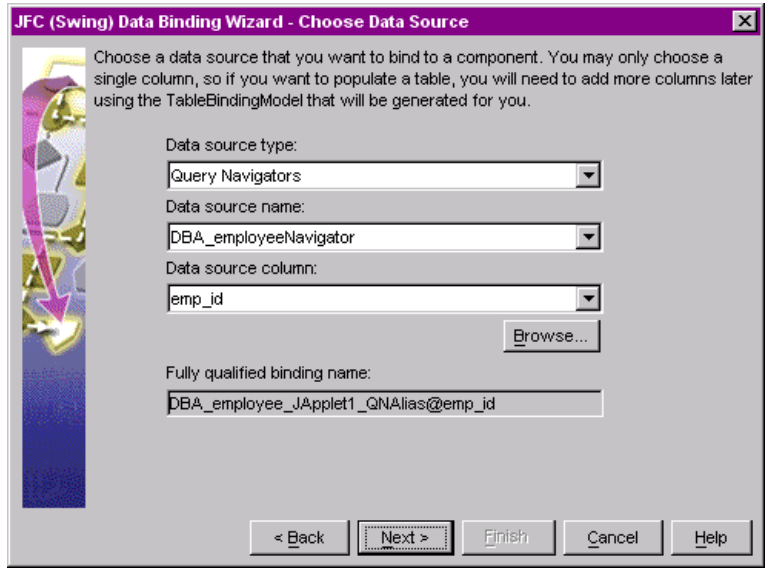
To continue, you'll have to click Cancel to exit the Add/Data Bind Component Wizard, and create a datasource for this project. For more information on creating a datasource, see "Identifying the datasource" on page 3-10. If you have a datasource for this project, the Choose Data Source page appears. The Choose Data Source page is pictured in the next illustration.

You can select only a single column to bind with a single component. If you want to add more columns, you'll need to add them using `TableBindingModel`. For more information on using `TableBindingModel`, see "Working with BindingModel and TableBindingModel" on page 4-25.

To select a datasource:

1. Select a datasource type from the datasource type drop-down menu.
2. Select an existing datasource name from the Data Source name drop-down menu.
3. Select a column from the Data Source column pull-down menu with which to bind to the selected component.

You can also use dcNAVIGATOR to help you visualize the available data sources. Click Browse to open dcNAVIGATOR. For more information, see "Using the Data Connection Navigator" on page 3-41.



4 Click Next to select the Swing component.

Selecting a Swing component

This page lets you select an existing Swing component or add a new one to the project. As you select from the list of available components, the text box at the bottom of the wizard page provides you with some information about that component.

To select an existing component:

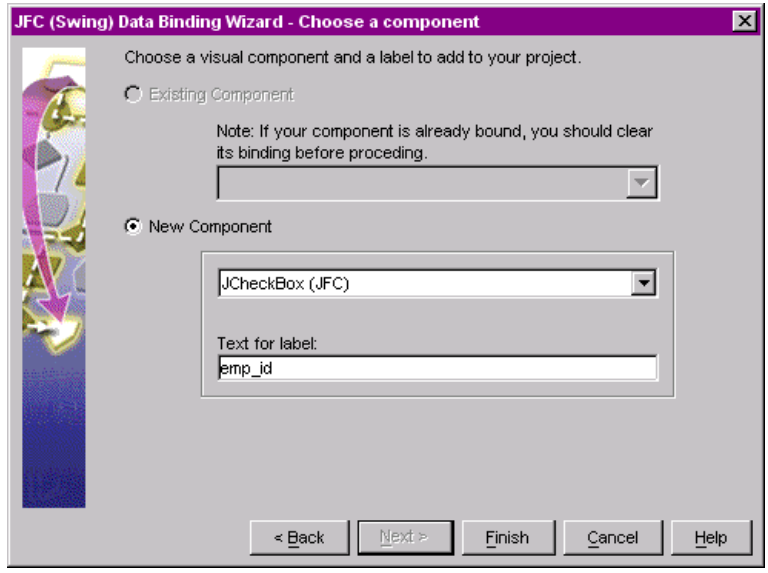
1 Click Existing Component.

2 Select an existing component from the drop-down menu.

3 Click Next to advance to the Summary wizard page.

To add a new Swing component:

1. Click New Component.
2. Select a Swing component from the drop-down menu.
3. Type in a label for the component you selected.



4. Click Finish to create the data binding.

   The component, label, and `BindingModel` component appear on your form.

Working with BindingModel and TableBindingModel

Binding data from a datasource to a Swing component is done using a binding model component. Visual Cafe provides two binding models: `BindingModel` and `TableBindingModel`. A `BindingModel` can be used with all components except `JTable`. A `TableBindingModel` is exclusively used to bind data to a `JTable` component.

Data sources you could bind to a component using a binding model include `Query Navigator`, `Stored Procedure Adapter`, and `Calculation Adapter`.

Note: You can drop a model compnent on a form before creating the component you'll bind it with.

These models have two important properties: `Data Binding` and `Component`.

The `Data Binding` property represents the main link to the component. This link contains only one row of data. It is the main link of a `TextField`, the selected item in a `TextArea`, or the current item in a `ComboBox`.

The `Component` property identifies the Swing component to which you want to bind the data model. For more information on data binding, see "Data binding in Visual Cafe" on page 4-12.

To set the BindingModel properties:

1. Drop a `BindingModel` or `TableBindingModel` component on the form.
2. Click the `Data Binding` property to create the datasource binding information.
3. Set the component property of the Swing component to which you want to bind data.

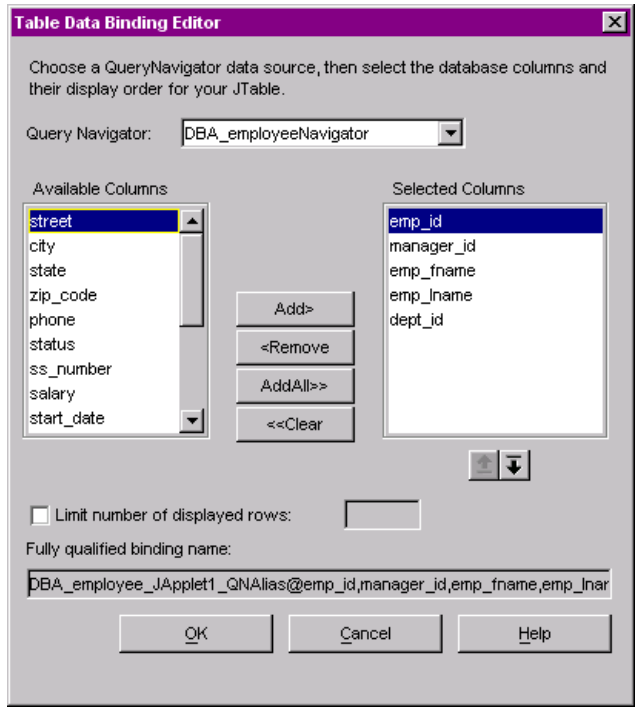
Table data binding

The Add/Data Bind Component Wizard is the easiest way to create a data binding between a `JTable` component and a datasource. However, the Add/Data Bind Component Wizard only allows you to bind one component per column. If you want to bind more than one column to a component, you use the Table Data Binding Editor.

To open the Table Data Binding Editor:

1. Select the `JTable` component in the Property List or Form Designer for which you want to create or modify a binding.
2. From the Property List, select `Model`, then select `new TableBindingModel`.

The Table Data Binding Editor appears:



3 Select the appropriate `Query Navigator` from the Query Navigator drop-down menu.

4 From the Available Columns window, select the appropriate columns and click Add.

The columns you select move to the Selected Columns list.

5 Change the order of the selected columns, if necessary, by clicking Up or Down.

6 Click Limit number of displayed rows if necessary, then specify the limitation.

7 Click OK to create the databound table.

The Table Data Binding Editor closes.

Dynamic List binding

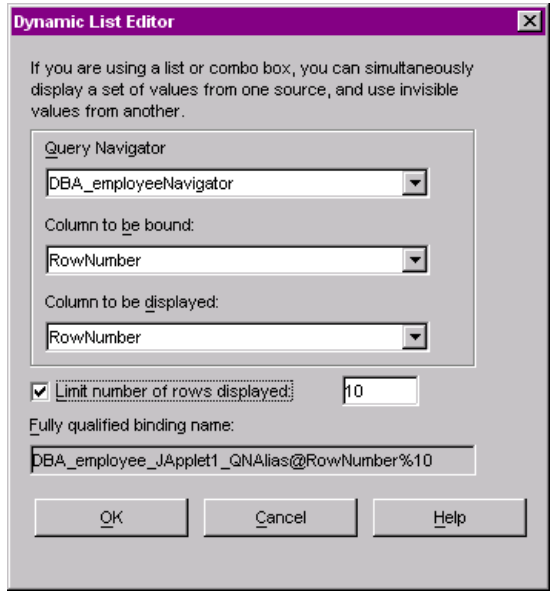
JLists and ComboBox components are designed for a user to see dynamic data and select from it. The Dynamic List Editor helps you to populate `JLists` and `JComboBoxes`.

To start the Dynamic List Editor:

1. Drag a `JList` or `JComboBox` component from the Swing tab of the Component Palette and drop it on an open form.
2. In the Property List, select the `Model` property for the `JList` or `JComboBox`.
3. In the `Model` property, click the field to the right and select `newBindingModel` from the drop-down list.

   A new `BindingModel` appears in the Project window.
4. In the Property List, select the new `BindingModel`.
5. Click the `Dynamic List` property.
6. Click on the ellipsis that appears to the right of the field.

The Dynamic List Editor appears:



7. Select the appropriate `Query Navigator` component from the `Query Navigator` drop-down menu.
8. Select the column to bind from the Column to be bound drop-down menu.
9. Select the column to display from the Column to be displayed drop-down menu.

   The fully qualified binding name appears.
10. If necessary, click Limit number of rows displayed to show only the first 10 rows (default).
11. Click OK to create the data binding.

C H A P T E R 5

Managing Data and Connections

After creating a databound project, you're ready to retrieve and store data, format it, and display it to the users of your program. This chapter contains information about the following topics:

- Creating SQL queries
- Using Query By Example
- Working with masks

Working with SQL

When you want to retrieve data from a database, you send a **query** to the database. This query specifies a request for information such as the names of employees who been employees three years or more.

One method for creating a query is by using **Structured Query Language (SQL)**. SQL is the standard language for creating, querying, and maintaining relational databases.

The standard SQL format for a query is:

```
SELECT X1, X2, X3, … from Y
```

where `X` represents a series of database columns that are separated by commas; and `Y` is the name of the database table. If the SQL string is not in this standard format, the database dialogs and wizards can't parse the information. You'll have to edit that string manually.

Visual Cafe Database Edition makes it easy for you to specify `SQL SELECT` statements by providing several dialog boxes to assist you. Alternatively, you can specify an SQL statement directly in the Java code.

A `SELECT` statement has these basic syntactical pieces, some of which are optional:

`SELECT` *columns* `FROM` *table* `WHERE` *filter* `ORDERBY` *sort-order*

You can add more SQL clauses to the end of a standard SQL string. Examples include: `<ORDER>`, `<BY>`, `<HAVING>`, `<LIKE>`, and `<WHERE>`.

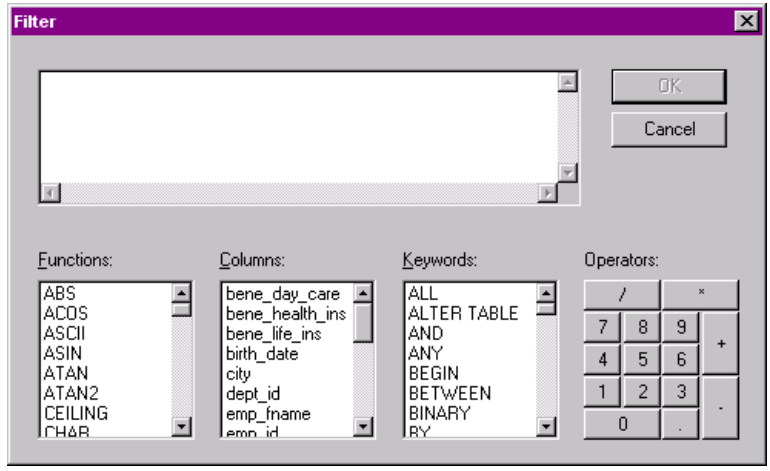
Defining SQL SELECT statements through properties

When you use a SQL SELECT statement (also called a **Select Clause**), *columns* and *table* are specified in the `Table Name` property of the `Record Definition` component. *Filter* is specified in the `Filter` property of the `Query Navigator` component. *Sort-order* is specified in the `Sort Order` property of the `Query Navigator` component.

To specify a custom WHERE clause:

1. In the Property List, select a `Query Navigator` that contains the Record Definition component you want to work with.
2. Click `Filter`.
3. Click on the ellipsis that appears to the right of the `Filter` property.

The Filter Editor appears:



4 Create your filter by selecting items from the following table:

Section	Description
Edit box	The SQL statement you are editing.
Functions	Functions you can add to the SQL statement. The functions in this list are the functions the database supports.
Columns	Database columns that are in the current `Query Navigator`. You can add any of these columns to the SQL statement.
Keywords	Keywords you can add to the SQL statement. The keywords in this list are the keywords the database supports.
Operators	Numbers and mathematical operations you can add to the SQL statement.

The SQL `Filter` appears in the text area as you make selections.

5 Click OK to close the Filter dialog box.

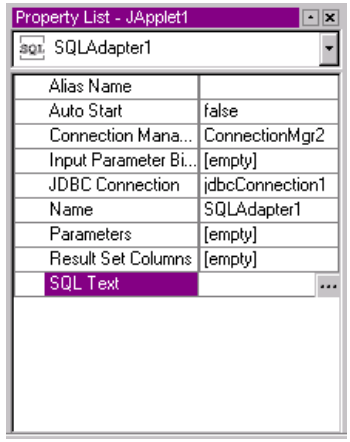
Using SQL Adapter components

The `SQL Adapter` component is designed to assist you in implementing SQL functionality in your projects. When you add a `SQL Adapter` component to your project, you set the properties for it in the Property List. The resulting source code becomes the framework for your SQL development.

To add a SQL Adapter to your project:

1 From the Data Source tab of the Component Palette, drag a `SQL Adapter` and drop it onto the form.

The properties for the `SQL Adapter` appear in the Property List:



2 Set the properties for the `SQL Adapter`.

Property	Description
`Alias Name`	The unique name given to the set of records managed by `Query Navigator`.
`Auto Start`	Determines if the `SQL Adapter` executes the SQL statement on applet startup.
`Connection Manager`	Manages all the connections of a program
`Input Parameter Binding`	Creates a databinding between input parameters and a datasource.
`Name`	The default name of the `SQL Adapter`.
`Parameters`	Opens the Paramters Editor to add, delete, and sort parameters.
`Result Set Columns`	A drop-down menu where you select result set columns.
`SQL Text`	Opens the SQL Text Editor to create SQL statements.

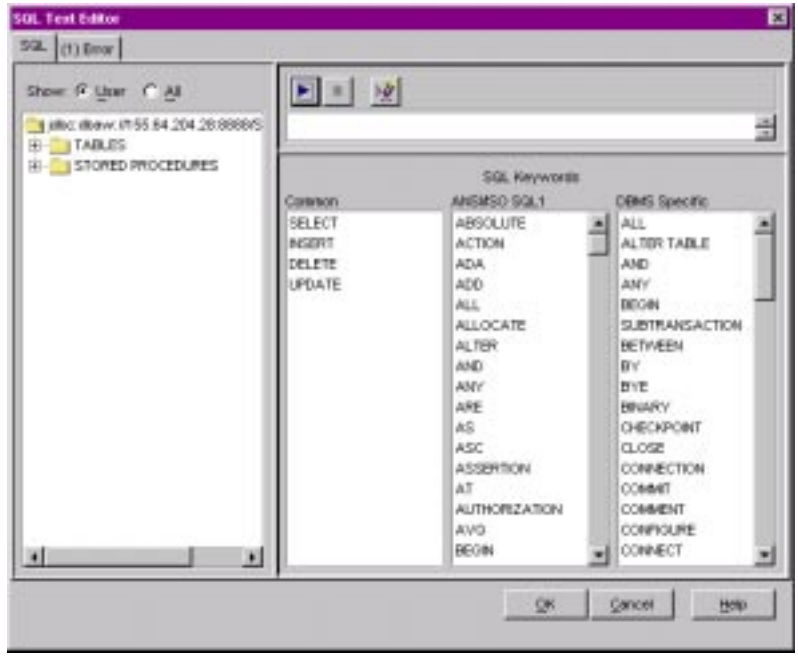
Using the SQL Text Editor

The SQL Text Editor enables you to create SQL queries by selecting the keywords for an SQL query. As you select the keywords for a query, the SQL Text Editor displays them in a scrollable text window. You can test your SQL query in this text window.

To open the SQL Text Editor:

1. Drag a SQL Adapter from the Data Source tab of the Component Palette and drop it on to the form.
2. In the Property List, click `SQL Text`.
3. Click on the ellipsis that appear.

The SQL Text Editor appears:



4. Create a SQL query by clicking keywords from the views.
5. Click Execute to test the SQL query.
6. Click OK to add the SQL statement to your project.

Working with Query By Example

Query By Example (QBE) is a model used to query relational data. QBE lets your application's users easily specify a query in a way that resembles data entry. Users search for a subset of records by filling out a form that represents an example of the type of record desired.

For example, if you enter `Jones` in a `Last Name` field, the query would return only the records that have Jones as the last name. If you enter `>1002` in an `Order Number` field, the query returns the records that have an order number greater than 1002. In SQL terms, this example is a conditional expression for the `WHERE` clause of the `SELECT` statement.

Creating QBE queries

You create QBE queries by entering values into the fields of a form. You can create more complex queries in QBE by combining values and operators. The following table shows all recognized operators and what they do in a field.

Operator	Meaning for text fields	Meaning for numeric fields
`=`	Exact match, Equal to	Equal to
`>`	Alphabetically follows text	Greater than
`<`	Alphabetically precedes text	Less than
`>=`	Greater than or equal to	Greater than or equal to
`<=`	Less than or equal to	Less than or equal to
`<>, !=`	Not equal to	Not equal to
`%, *`	Variable-length wildcards	N/A
`_, ?`	Single-character wildcards	N/A
`/, !`	Boolean `NOT`	Boolean `NOT`
`And, &`	Boolean `AND, &`	Boolean `AND`
`Or, |`	Boolean `OR, |`	Boolean `OR`
`\`	Escape character	N/A
`..`	Equal-to range including start and end items	Equal-to range including start and end items

Note: The escape character (\) allows literal inclusion of text normally considered a wildcard (the `%`, `*`, `_`, and `?` characters).

You can place operators in multiple fields in a form to create more focused queries. The following examples assume you're looking at a form with a last name text field.

- To find employees whose name starts with `s` but are not named `Smith`, enter:

  ```
  s*&!smith
  ```

- To find an employee whose last name might start with an s or a c enter:

  ```
  s*|c*
  ```

- To find a text item that uses an asterisk (`*`), enter:

  ```
  s\**
  ```

- To find an item whose name start with `s` and next character is (`%`), enter:

  ```
  s\%%
  ```

- To find an item by excluding others, enter:

  ```
  <>smith or !=Jones
  ```

- To locate everyone whose name starts with `a` through `d`, enter:

  ```
  aaphman..dzerba
  ```

The following examples assume you're looking at a form with an employee ID numeric field.

- To find employees whose number starts with 1 but are not 10,100, or 1000, enter:

  ```
  >=1!10AND!100AND!1000
  ```

- To locate employee IDs that are between 1 and 2000, including 1 and 2000, enter:

  ```
  1..2000
  ```

The following examples assume you're looking at a form with a quantity numeric field.

- To locate all quantities greater than 5, enter:

  ```
  >5
  ```

- To locate all quantities not equal to 5, enter:

  ```
  <>5 or !=5
  ```

To create a query:

1 In the running applet, click QBE.

The `Query Navigator` is positioned on the QBE row and places the form into QBE mode. The `RecordNumber` and `RecordState` fields display QBE whenever the form is in QBE mode.

2 Click Retrieve to perform the query.

Each time you click Restart, the `WHERE` clause takes into account the QBE values.

To modify a query:

1 In the running applet, enter QBE mode by clicking the QBE button.

2 Modify the form's fields with the new query values.

3 Click Retrieve to perform the query.

If you change some records in your database and click QBE, a dialog box displays, asking you to either Save, Discard, or Cancel the previous changes. Click the appropriate button and continue.

To re-initialize the form for a standard query:

1 In the running applet, enter QBE mode by clicking the QBE button.

2 Delete the contents of the fields.

Clicking Undo while the `RecordNumber` and `RecordState` fields are displayed also clears any data on the form.

Using QBE with master-detail relationships

To use QBE in a detail view, you must bind a QBE button to the detail view, using the `Query Navigator`'s `startQBE()` method.

If you change some records in the detail view and click the master view's QBE button, a dialog box displays with the following message:

```
You made some changes in the details.
```

You're offered three choices: Save, Discard, or Cancel. Click the appropriate button and continue. If you change some records in the master view and click the master view's QBE button, a dialog box displays with the following message:

```
You made some changes in the master's records
```

You're offered three choices: Save, Discard, or Cancel. Choose the appropriate button and continue.

If you modify the join value in the details (in the QBE row), the join value is added to the `WHERE` clause. For example, if the initial join clause is:

```
WHERE Department.depart_id=Employee.depart_id - QBE
value:Employee.depart_id > 100
```

the `WHERE` clause changes to:

```
WHERE (Department.depart_id=Employee.depart_id) AND
(Employee.depart_id > 100)
```

In this example, the `WHERE` clause is nonsense; the result is always null. This example is true when the initial join clause is `"!="`(not equal) instead of `"="` (equal).

Note: Restarting the master view takes into account the detail view's QBE values.

Predefining a QBE statement

To enter default QBE values before running an application, write the following code in the `init()` method of your applet:

```
init(){
.....
Query Navigator.setValueAsQBE(value,0,db_column_number); ....
}
```

This technique is useful for placing examples in the QBE fields, such as how to use QBE grammar, or for having a predefined QBE statement.

You can also predefine QBE statements at run time.

Using the QBE WHERE clause

To view the final `WHERE` clause including the QBE values, you must use the `Query Navigator.getWhereClause()`.

When working with an applet, you can display the `WHERE` clause by setting applet property of the `Query Navigator` in the `init` method of your applet:

```
init(){
....
Query Navigator.setApplet(this);
....
}
```

Working with field masks

Masks are used to restrict the type of data a user can enter in a component. You can create **field masks** to help your program's users know what kind of data should go into an input field.

At application design time, a field mask acts as another property of a databound component. As such, it can be edited like any other property. The components that have editable masking properties are:

- `MaskedTextField`
- `DBMaskedTextField`
- `JMaskedTextField`

You edit these properties in the `Mask` property field of the selected component.

The following components also have a mask property called `Currency`:

- `CurrencyTextField`
- `DBCurrencyTextFIeld`
- `JCurrencyTextField`

The `Mask` property of the `FormattedTextField` component does not support the field specification language and contains its own predefined set of values for accepting input. For more information about the `FormattedTextField`, see the Online Help.

Input masking versus data masking

Input masking is different from "presentation formatting" and "data masking" that is for translating user input into storage format. In fact, data masking and presentation formatting are mutually exclusive, with data masking taking precedence when both are defined in the same field.

Input masking is in control during the entire life cycle of the field-editing process. By contrast, formatting is a translation operation performed once on the data value before it's placed in the field display component.

Masking obtains control when a component receives focus to display the data (and/or the mask literals). It then retains control during each keystroke in order to assure a match between the input character and the relevant filter for the current cursor position. It also updates the cursor position in order to skip literals. Masking also obtains control on a lost-focus (when an object no longer has focus) event when it strips mask literals from the field before committing the value to the data store.

Using the field-masking language

The field-masking language enables you to specify what type of input is allowed at each character position of the data input string. In addition to input filtering, this field-masking language allows the field to be displayed to the users of your program before any data is entered in the field.

The field-masking language consists of two parts: syntax and lexicon.

Syntax

In the filter-masking language, a sentence is any string of tokens, that is, keywords and literals. The order of the tokens doesn't matter syntactically. In other words, the appearance of token `X` in a string in no way restricts the set of possible tokens that follow token `X`. Basically, this means there are no syntax rules in this language.

Lexicon

The lexical tokens of the language are all one character long. If a token is not a keyword – which in this case is a character or set of characters (not a word) – it's a literal.

The lexical portion of the masking language consists of three types of lexical elements (all one character in length):

- **Filters**, which are characters that filter input at a specific position.
- **Commands**, which carry out operations on a subfield of input. characters
- **Literals**, which appear unaltered in the field at run time.

Filter characters

The filter characters are described in the following table:

Filters	Description
`0`	Digits: (0 through 9, followed by required keyboard entry; plus [+] and minus [-] signs not allowed)
`9`	Digit or space (entry not required; plus and minus signs not allowed)
`#`	Digit or space (entry not required; blank positions converted to spaces, plus and minus signs allowed)
`L`	Letter (A through Z, entry required)
`?`	Letter (A through Z, entry optional)
`A`	Letter or digit (entry required)
`a`	Letter or digit (entry optional)
`&`	Any character or a space (entry required)
`C`	Any character or a space (entry optional)

Command characters

The command characters are described in the following table.

Command	Description
`.`	Decimal place holder
`,`	Thousands separator (for example, 45,012)
`: ; - /`	Date and time separators
<	Causes all characters that follow to be converted to lowercase
>	Causes all characters that follow to be converted to uppercase
\	Escape character: causes the character that follows to be displayed literally (for example `\z` displays z on screen)

Literals

A character in a field mask specification is automatically a literal if it is neither a filter nor a command. Literals are typically used as punctuation symbols that separate subfields of the data (for example, dashes in a Social Security Number field) or as descriptive information about the characters they precede or follow (for example, currency symbols).

Specifying field masks at runtime

The field masking language is invoked at run time on an **insert** or **update** operation. Insert refers to the editing of a new record, that is, the masked field has no data at insert (edit mask) time. Update refers to updating existing data under mask control.

Insert

There are three conditions where edit masking is invoked during an insert operation. These conditions are:

- when editing starts
- during editing
- when editing ends

When editing starts

The component displays text consisting of the mask literals in their proper positions, and underscore characters in each position represented in the mask by a filter. The cursor is then placed at the position of the first filter in the mask. This differs from Microsoft Access, which selects the entire field and places the cursor at the first position, even if that position is a literal. The first acceptable character then goes into the position of the first filter.

An exception to this setup is that numeric-type fields are initialized to zero. A single, zero character is displayed in the rightmost digit filter position. In addition, if the mask contains a decimal point literal, each digit filter position to the right of the decimal point is also displayed as zero.

During editing

The cursor is restricted to those positions in the field that correspond with filters in the mask. Only characters that pass the filter are accepted. The cursor remains fixed until an acceptable character is typed or a navigational event occurs (including cursor control key strokes).

If the mask contains the decimal point literal and the cursor is to the left of the decimal point, then if the user types the decimal point character, the cursor is moved to the first filter position to the right of the decimal point.

When editing ends

Except for the decimal point literal in numeric fields and date/time separators in date or time fields, all literals are stripped from the field contents before being passed to data storage.

Update

There are three conditions where edit masking is invoked during an update operation. These conditions are:

- when editing starts
- during editing
- when editing ends

When editing starts

The characters in the data value are displayed in the format given by the mask. Each filter position, starting from the left of the mask and proceeding to the right, is filled with a character from the data value string, also starting from the left. This process is trickier than it sounds because the string coming from data storage may include punctuation symbols in positions where the mask has literals that have the same meaning. Such data characters are skipped in the filling process. For example, if a numeric field has a mask of `00.00` and is being filled with a data string `12.34`, the decimal point in the data string is skipped. After setting the text in the field, the cursor is placed at the position of the first filter.

During editing

The cursor is restricted to those positions in the field that correspond to filters in the mask. Only characters that pass the filter are accepted. The cursor remains fixed until an acceptable character is typed or a navigational event occurs (including cursor-control keystrokes). If the mask contains the decimal point literal and the cursor is to the left of the decimal point, then if the user types the decimal point character the cursor is moved to the first filter position to the right of the decimal point. This behavior is the same regardless of the state of the keyboard's Insert key.

When editing ends

Except for the decimal point literal in numeric fields and date/time separators in date or time fields, all literals are stripped from the field contents before being passed to data storage.

APPENDIX A

Understanding the DataBus

The DataBus is a low-level API/architecture for Visual Cafe. It enables components (JavaBeans) to share data with other components. The DataBus interface enables mediators to bind components to datasources.

The DataBus architecture also allows you to easily create applications that use JavaBeans that exchange data asynchronously. This data exchange can be done with applets, with beans that are created in a builder environment such as Visual Cafe. The DataBus is also used by other Java classes and programs, such as applications, servlets, and so on.

How data flows between components

The DataBus helps components work together within the same Java Virtual Machine (JVM). The current design of the DataBus doesn't directly deal with components working in separate JVMs, such as those on different processors, though it does contain features that allows you to build a bridge between JVMs.

In general, Java Beans that are loaded from a single class can "see" other beans from the same loader and can make direct method calls on those beans. However, these cross-bean calls are currently based on well-known interfaces or base classes. Beans use **introspection** to learn or discover information about peer beans at run time. One bean can learn about an API supported by another by detecting certain design patterns in the names of the methods that are discovered through introspection. By contrast, the DataBus interfaces create a tightly typed contract between cooperating beans. No inferring is required, and procedure calls are direct.

The DataBus interfaces allow you to create data flows between cooperating beans. In contrast to more traditional event/response models, the DataBus interfaces have very few events and have a fixed set of method calls for all components. These interfaces interpret the contents of data that flows across the DataBus, not the names or parameters from events, nor in the names or parameters of callbacks.

APPENDIX B

Creating Custom Calculation and Validation Rules

Anticipating the different types of calculation and validation rules that might be used in applications is a complex task. Visual Cafe Database Edition provides a predefined set of these rules are described in this chapter. For example, one such pre-defined rule included with Visual Cafe is a `MultiplicationFunction` rule. This rule returns the product of one or more columns. A typical use for this rule would be in a *product* table and to display the value of *quantity * price*.

If none of Visual Cafe's predefined rules apply to a particular task in your program, you can take advantage of Visual Cafe's flexible architecture to define your own custom rules.

About developing custom rules

Developing custom rules involves implementing `Calculation` and `Validation` interfaces that are included in the `symantec.itools.db.beans.brl` package.

The `Calculation` and `Validation` interfaces are:

- `ComputedColumn`
- `Calculation`
- `Validation`

The following section describes the interfaces for creating and implementing your own rules in Visual Cafe.

ComputedColumn interface

The main interface for both calculation and validation functionality. It implements the `getDisplayString()` method, which returns a string that describes the function of the rule.

Calculation interface

This interface contains the following methods:

- `getComputedValue (DerivationRow inputRow, int outputDataType)`

  The method that performs the calculation. The `Calculation Adapter` calls this method every time a calculation is performed. The `Calculation Adapter` is responsible for constructing `DerivationRow` which contains all the inputs to the rule with the values. The `outputDataType` represents the `java.sql.Types` value which is used for representing the output data type.

- `isUsedForAggregate()`

  Returns a true or false value depending on whether the calculation can be applied to a set of rows. If this method returns a true value, the calculation rule is applied to every row. When this method returns a false value and the user wants to apply this method to a set of rows, an error message displays.

Validation interface

This interface contains the following methods:

```
validate(ValidationRow inputRow)
          throws ValidationException
```

The method which actually performs the validation. The `Validation Adapter` calls this method to perform the validation. The `Validation Adapter` constructs `ValidationRow`. `ValidationRow` contains the input values for the rule. If the validation condition is not satisfied, then a `ValidationException` is thrown.

Sample custom calculation rule

As an example for implementing a custom calculation rule, you could develop a calculation rule that calculates simple interest. This rule would need variables that represent a corresponding principal amount (`P`), number of years (`N`) and the rate of interest (`R`) using the formula `(P*N*R)/100`.

```
package interest.calculation;
import symantec.itools.db.beans.brl.Calculation;
 import symantec.itools.db.beans.brl.DerivationRow;
 import symantec.itools.db.beans.brl.CompColInputMember;
/**
  * Class which can be used to calculate simple interest
  * for a specified prinicipal amount[P], number of years [N]
  * and rate of interest [ R ]. The formula used is to
  * calculate the interest is P*N*R/100
  *
  */
public class SimpleInterestCalculation implements Calculation {
 /**
  * Method to return a String which would
  * be used for displaying method information to the user
  * Example: For a class which calculates the sum of three
  * columns
  * using a description like "sum( ?,?,? )" would be appropriate.
  *
  * @return String representing the display name
  */
    public String getDisplayString() {
    return "This method is used calculate to simple interest";
    }

    /**
     * return the computed value from the passed in input Row
     * The first parameter ,ComputeRow, is a grouping of columns
     * which are to be used for calculating the new derived
     * value.These columns could represent constant as well as
     * real database columns.

     * @param inputRow row of input columns
     * @param outputDataType datatype for desired output as in
     * java.sql.Types
     * @return Object representing the new computed value
     * @see ComputeRow
     * @see CompColMember
     */
    public Object getComputedValue( DerivationRow inputRow,
                                          int outputDataType )
 {
    // To make the implementation simpler, the following
    // assumptions are made:
    // There are 3 inputs to the calculation:
    // 1st input: Principal amount and data type is integer
    // 2nd input: Number of years and data type is integer
    // 3rd input: Interest rate and data type is integer
    // Desired data type for outputDataType is:
```

```
    // java.sql.Types.DOUBLE
       Object retValue = null;

        try {
            if (inputRow.getNumberOfSourceColumns() == 3) {

               CompColInputMember amountMember =
               inputRow.getSourceColumnMember( 1 );

               CompColInputMember numYearsMember =
               inputRow.getSourceColumnMember( 2 );
               CompColInputMember rateIntMember =
               inputRow.getSourceColumnMember( 3 );

               int amount = new Integer(
               amountMember.getValue().toString() ).intValue();

               int years = new Integer
               (numYearsMember.getValue().toString()).intValue();

               int rate = new Integer
               (rateIntMember.getValue().toString()).intValue();

               retValue = new Double( (amount*years*rate)/100 );
            }
        }
        catch (Exception e) {
           System.out.println( "Exception in calculating
           interest: " + e.getMessage() );
        }

        return retValue;
    }

    /**
     *Method to determine if this can be used for aggregate
     * calculations
     *
     * @return true or false
     */
    public boolean isUsedForAggregate() {
           return false;
    }
}
```

Sample Code for Calculation rule bean info

The `Calculation Adapter` allows you to import a custom rule and add it to the Component Library. Once it is in the Component Library, you can use it again. The Add Calculation Adapter, by default, selects the classes that implement the `Calculation` interface from the `Calculation Formulas` folder. Creating a `bean info` method for this class with this folder name ensures that it is added to the appropriate folder in the Component Library.

```
package interest.calculation;

import java.beans.SimpleBeanInfo;
import java.beans.BeanDescriptor;
import symantec.itools.beans.SymantecBeanDescriptor;

/**
 * BeanInfo for SimpleInterestCalculation
 *
 */
public class SimpleInterestCalculationBeanInfo extends
 SimpleBeanInfo {
    public BeanDescriptor getBeanDescriptor() {
        SymantecBeanDescriptor desc = new SymantecBeanDescriptor(
 SimpleInterestCalculation.class );
        desc.setFolder( "Calculation Formulas" );

        return desc;
    }
}
```

Custom validation rules

Developing a custom validation rule is very similar to developing a custom calculation rule with the only difference being that the `Validation` interface has to be implemented instead of the `Calculation` interface.

Predefined calculation rules

This table describes the predefined calculation rules that are part of Visual Cafe Database Edition.

Name	Description
`CurrentTimeFunction`	Returns the current time.
`CurrentDateFunction`	Returns the current date.

Name	Description
`MinFunction`	Finds minimum value from a set of inputs. It applies only to number data types.
`MaxFunction`	Finds maximum value from a set of inputs. It applies only to numeric data types.
`MultiplicationFunction`	Returns product of a set of inputs. It applies only to numeric data types.
`SubtractFunction`	Returns difference from a set of inputs. It applies only to numeric data types.
`StringConcatFunction`	Returns concatenated string from a set of inputs. Applies only to inputs of data types which can be represented as string.
`AdditionFunction`	Returns sum from a set of inputs. It applies only to numeric data types.
`DateSubtractFunction`	Subtracts an integer value from a date.
`DivisionFunction`	Performs division operation for a set of inputs. It applies only to numeric data types.
`DateAddFunction`	Adds an integer value to a date.

Pre-defined validation rules

This table describes the predefined validation rules that are part of Visual Cafe Database Edition.

Name	Operators	Description
`StringValidation`	`Equals`	Compares two strings for equality
	`Contains`	Compares strings and determines if the second string is present in first string.
`NumberValidation`	`>`	Greater than comparision
	`<`	Lesser than comparision
	`>=`	Greater-or-equal-to comparision
	`<=`	Lesser-or-equal-to comparision
	`!=`	Not equal to comparision
	`==`	Equality comparision
`DateValidation`	`Equals`	Check two dates for equality
	`Before`	Date1 is less than date2
	`After`	Date1 is greater than date2

Note: All validation rules assume the format:
{*Column1*} {*operator*} {*Column2*}

A P P E N D I X

C

Using dbANYWHERE Components

This section describes the dbANYWHERE components that are used to connect to the dbANYWHERE Server. These components include:

- `Session`
- `ConnectionInfo`
- `RelationViewPlus`

Working with the Session component

The (non-visual) `Session` component defines the connection to a dbANYWHERE server through a TCP network connection, and provides access to the dbANYWHERE classes. At run time, the Session component generates code that creates a session with the dbANYWHERE server.

Add a `Session` component prior to adding the `RelationViewPlus` component. Use the name of this component as the value for the `RelationViewPlus` component's `Session` property. The easiest way to add these components is by using the Databound Project Wizard and Add Database Table Wizard.

- Access the Project Wizard by first selecting New Project from the File menu. Then from the New Project window, open the DataBound Project Wizard.
- Access the Add Database Table Wizard by selecting the Database menu and choosing Add Database Table, or by selecting the component from the Component Palette or Component Library.

One `Session` component represents only one URL, so you can add multiple `Session` components to show data from different dbANYWHERE servers.

Note: It is recommended that all connections to a single dbANYWHERE server use the same `Session` component. This helps to reduce the number of server connections.

Properties

The `URL` property defines the dbANYWHERE server's URL (location and port number).

The `Name` property defines the name of the dbANYWHERE server that the project expects to access.

Note: Do not change `Session` component properties at run time.

Working with the ConnectionInfo component

The `ConnectionInfo` component is a non-visual component that defines a datasource. Connection information includes the name of the data source, user login name, and user password.

You can drag the `ConnectionInfo` component from the Component Palette or Component Library onto a form at any location, or into the Project window. Also, you can use the DataBound Project Wizard or the Add Database Table Wizard. These two wizards create any necessary `ConnectionInfo`, `Session`, and `RelationViewPlus` components.

A project can contain more than one `ConnectionInfo` component. Each datasource has one `ConnectionInfo` component. The `ConnectionInfo` component must have a `Session` component to connect it to the dbANYWHERE server.

Properties

The `ConnectionInfo` component contains the following properties:

- Auto Disconnect
  Controls automatic disconnection from the database when you close a `MultiView`.
- `Data Source Name`
  Indentifies the datasource to connect to.
- `Name`
  Contains the name of the actual Java object that instantiates the `ConnectionInfo` class. This value usually closely matches the data source name. Visual Cafe replaces characters, such as spaces or punctuation, in the datasource name with underscores to create a valid Java variable name.
- `User Name`
  Provides logon requirements.
- `User Password`
  Provides logon requirements.

Note: Do not change `ConnectionInfo` component properties at run time.

Working with the RelationViewPlus comopnent

The `RelationViewPlus` component is a non-visual component that defines a view of a database table and defines how the data is retrieved form the dbANYWHERE server — when a query is made against the database.

The easiest way to add a `RelationViewPlus` component is by using the Databound Project Wizard and Add Database Table Wizard.

- Access the Project Wizard by first selecting New Project from the File menu. Then from the New Project window, open the DataBound Project Wizard.

- Access the Add Database Table Wizard by selecting the Database menu and choosing Add Database Table, or by selecting the component from the Component Palette or Component Library.

Note: You must add the `RelationViewPlus` component after you add the `Session` and `ConnectionInfo` components. The easiest way to add these components is by using the Databound Project Wizard and Add Database Table Wizard.

A project can contain multiple `RelationView` components. These views may have a joined relationship, or be independent. In the case of a join, the first `RelationViewPlus` in a project is used as a master view. When you add a second `RelationViewPlus` component, you create a detail view, thereby forming a master-detail relationship.

Use the DataBound Access Project Wizard for your first `RelationViewPlus` (the master view) and then Add Database Table Wizard for additional views.

When a master-detail relationship is formed, the relationship is preserved across a transaction. When the master view is saved, the detail view is also saved, and the entire transaction is committed. Similarly, when the detail view is saved, the master view is also saved.

Using the RelationViewPlus and MultiView classes

`MultiView` is a container class for the dbANYWHERE database transaction object. A `MultiView` object contains a transaction's `RelationViewPlus` component. To interact with a dbANYWHERE Server, you do not have to interface directly with a `MultiView` object — the `RelationViewPlus` class can encapsulate a `MultiView` object. For example, `RelationViewPlus.saveMultiView( )` translates to `MultiViewPlus.save( )` for the `RelationViewPlus` component's parent `MultiView`. However, a `MultiView` object can be useful to global transactions that have a handle to a `MultiView` object. For more information on the `MultiView` object, see the dbANYWHERE API online HTML reference.

Properties

A `RelationViewPlus` component's SQL statement (`Select Clause` property) defines the view. You can create a view that is a detail view or one that isn't.

- `ConnectionInfo`

  The name of `ConnectionInfo` component that the `RelationViewPlus` is based on.

- `Initial Record Position`

  Two possible values: `first` or `new`.

- Join
  - `Parent`: Names the parent in the join relationship.
  - `Join Columns`: Invokes the Join Definition Editor.
  - `Cardinality`: Establishes the relationship between master and detail as one to one, many to many, many to one, or one to many.

- Name

  Unique name for this object.

- Optimistic Concurrency

  Specifies how to update the database when more than one client is accessing it. Optimistic concurrency allows muti-user access to data while maintaining data integrity and preventing update losses. Allowable values are: `All`, `Unique`, `Unique Modified`.

- `Select Clause`

  Invokes the SQL statement editor.

- `Session`

  The name of the `Session` object the `RelationViewPlus` is based on.

Note: Do not change `RelationViewPlus` component properties at run time.

A P P E N D I X

D

Working with Data Types

The following tables show mappings between data types in some common databases and JDBC. For more information, refer to the JDBC specification for the mapping between SQL types and Java types.

Informix

SQL data type	Description	JDBC mapping
BYTE	Stores any kind of binary data	LONGVARBINARY
CHAR(n)	Stores single-byte or multibyte sequences of characters, including letters, numbers, and symbols; collation is code-set dependent	CHAR
CHARACTER(n)	A synonym for CHAR	CHAR
DATE	Stores calendar date	DATE
DATETIME	Stores calendar date combined with time of day	TIMESTAMP
DEC	A synonym for DECIMAL	DECIMAL
DECIMAL	Stores numbers with definable scale and precision	DECIMAL
DOUBLE PRECISION	Behaves the same way as FLOAT	FLOAT
FLOAT(n)	Stores double-precision floating-point numbers corresponding to the double data type in C	FLOAT
INT	A synonym for INTEGER	INTEGER

SQL data type	Description	JDBC mapping
INTEGER	Stores whole numbers from -2,147,483,647 to +2,147,483,647	INTEGER
INTERVAL	Stores span of time	VARCHAR
MONEY(p,s)	Stores currency amount	DECIMAL
NCHAR(n)	Stores single-byte and multibyte sequences of characters, including letters, numbers, and symbols; collation is locale-dependent	CHAR
NUMERIC(p,s)	A synonym for DECIMAL	NUMERIC
NVARCHAR(m,r)	Stores single-byte and multibyte sequences of characters, including letters, numbers, and symbols of varying length; collation is locale-dependent	VARCHAR
REAL	A synonym for SMALLFLOAT	FLOAT
SERIAL	Stores sequential integers	INTEGER
SMALLFLOAT	Stores single-precision floating-point numbers corresponding to the float data type in C	FLOAT
SMALLINT	Stores whole numbers from -32,767 to +32,767	SMALLINT
TEXT	Stores any kind of text data	LONGVARCHAR
VARCHAR(m,r)	Stores multibyte strings of letters, numbers, and symbols of varying length; collation is code-set dependent	VARCHAR

Oracle

SQL data type	JDBC mapping
VARCHAR2(size)	VARCHAR
NUMBER(p,s)	FLOAT, DECIMAL If scale is zero, it will map to FLOAT. If scale is non-zero, it will map to DECIMAL.
LONG	LONGVARCHAR
DATE	TIMESTAMP
RAW(size)	BINARY
LONG RAW	LONGVARBINARY
ROWID	BINARY
CHAR(size)	CHAR
MLSLABEL	BINARY

Microsoft SQL Server

SQL data type	JDBC mapping
`binary (n)`	`BINARY`
`varbinary (n)`	`VARBINARY`
`char (n)`	`CHAR`
`varchar(n)`	`VARCHAR`
`datetime`	`TIMESTAMP`
`smalldatetime`	`DATE`
`decimal (p,s)`	`DECIMAL`
`numeric (p,s)`	`NUMERIC`
`float (n)`	`FLOAT`
`real`	`REAL`
`int`	`INTEGER`
`smallint`	`SMALLINT`
`tinyint`	`TINYINT`
`money`	`DECIMAL`
`smallmoney`	`DECIMAL`
`bit`	`BIT`
`timestamp`	`BINARY`
`text`	`LONGVARCHAR`
`image`	`LONGVARBINARY`

Sybase SQL Server

SQL data type	JDBC mapping
binary (n)	BINARY
varbinary (n)	VARBINARY
char (n)	CHAR
varchar(n)	VARCHAR
datetime	TIMESTAMP
smalldatetime	DATE
decimal (p,s)	DECIMAL
numeric (p,s)	NUMERIC
float (n)	FLOAT
real	REAL
int	INTEGER
smallint	SMALLINT
tinyint	TINYINT
money	DECIMAL
smallmoney	DECIMAL
bit	BIT
timestamp	BINARY
text	LONGVARCHAR
image	LONGVARBINARY

Watcom/Sybase SQL AnyWhere

SQL data type	JDBC mapping
`char(size)`	`CHAR`
`character(size)`	`CHAR`
`varchar(size)`	`VARCHAR`
`character varying (size)`	`VARCHAR`
`long varchar`	`LONGVARCHAR`
`int, integer`	`INTEGER`
`smallint`	`SMALLINT`
`decimal (p,s),`	`DECIMAL`
`numeric (p,s)`	`NUMERIC`
`float, real` (single precision floating point)	FLOAT, REAL
`double` (double precision floating point)	DOUBLE
`date`	DATE
`timestamp`	TIMESTAMP
`time`	TIME
`binary` (size)	BINARY
`long binary`	LONGVARBINARY

Microsoft Access

SQL data type	JDBC mapping
`Bit`	`BIT`
`Byte`	`TINYINT`
`Binary`	`BINARY`

SQL data type	JDBC mapping
Counter (auto-increment)	INTEGER
Currency	DECIMAL
Datetime	TIMESTAMP
Single (single precision floating point)	FLOAT
Double (double precision floating point)	DOUBLE
Short	SMALLINT
Long	INTEGER
Long Text	LONGVARCHAR
Long Binary	LONGVARBINARY
Text	CHAR

G L O S S A R Y

A

action

Action performed on a database:

- delete: Deletes the current record from the datasource.
- first: Moves the database cursor to the first record in the database table.
- new: Creates a new record in the database table.
- next: Moves the database cursor to the next record in the database table.
- prev: Moves the database cursor to the previous record in the database table.

adapter

Classes that implement an interface. The java.awt.event package provides, as a convenience, a series of listener classes that can be implemented by classes. You use adapters to work with business rules and stored procedures in a Visual Cafe project.

B

bean

A component complying with the JavaBeans standard. Also called a JavaBean. A bean is a reusable component that can be visually manipulated in a builder tool, such as Visual Cafe. The minimum requirements are that it can be instantiated (it is not an abstract class or interface) and has a class constructor method that takes zero parameters (it has a null constructor). It can also have properties and events, implement the serializable or externalizable interface, and follow method signature rules so it can be introspected.

boolean expression

Expression that evaluates to true or false.

C

Calculation Adapter

An adapter that uses a wizard to guide you through creating a calculation which is displayed in a text based component. Available math functions include addition, subtraction, multiplication, and sums.

client machine

Machine or program that requests and receives data from a server.

client software

Software for communicating with a database engine such as:

- Oracle Server client software
- Microsoft SQL Server client software
- ODBC32 Administrator

column

Part of a database table. A column represents a characteristic of the subject of the table. For example, a table that stores names and addresses has columns for the first name, last name, street number, street name, city, state, and zip code.

ConnectionInfo

When you use a databound wizard or drag a datasource to a project, Visual Cafe Database Edition creates a `ConnectionInfo` object. It shows the databases to which you are connected and includes the default user name and password, which you can remove. The `ConnectionInfo` object supports multiple queries and updates simultaneously, thereby reducing database resource requirements.

ConnectionManager

A bean used as a logical container for `JdbcConnection` beans. Dragging a datasource item from dbNAVIGATOR to a project creates `ConnectionManager` and `JdbcConnection` objects, which manage connections to a datasource. You have one `ConnectionManager` object per project, and one `JdbcConnection` object per datasource. You can also drag `ConnectionManager` objects to the Component Library.

constant

Data that does not change. Also called persistant data. For example, in the formula x=y+5, 5 is a constant since there is no way to assign a new value to 5.

context menu

Menu that appears when you click the secondary mouse button in a Visual Cafe window. A context menu is often called a pop-up menu.

custom code

Java code that Visual Cafe has not automatically created for you.

D

data binding

The mechanism by which data is automatically passed between components and the datasource. There are at least three ways of binding data in Visual Cafe: through a `BindingModel` and a Swing component; through a `mediator` and a standard AWT component, or directly from the datasource to a databound component.

databound

Components and properties that do not need additional coding to work with databases. Formerly, those were referred to as dbAWARE. Databound components contain properties that allow you to bind the component to a database row, column, table, or result set. You may also bind to a `Stored Procedure`, `Calculation`, or `Validation Adapter`.

datasource

Contains the information for creating a database connection, like the name of the database, its server, and its network location. A datasource is what identifies a database to an ODBC or JDBC-compliant database application. Before you can use your database in a Visual Cafe project, you must create its datasource. Sometimes a datasource is called a DSN (datasource Name).

dbANYWHERE API

An API developed by Symantec for accessing databases using the dbANYWHERE Server. It is an SQL-level database protocol that makes use of the extensions provided by dbANYWHERE.

dbANYWHERE Server	A database connectivity server that provides an implementation of the JavaSoft JDBC API as well as its own Java database connectivity API, called the dbANYWHERE API.
dbNAVIGATOR	Browser window which shows the servers, datasources, and contents of the datasources connected to those servers. You can also use the dbNAVIGATOR to perform drag-and-drop operations on your forms and components.
dcNAVIGATOR	Also called the Data Connection Navigator. A Browser window which displays all the current connections in use. Used to select a data connection object from those currently available on the Data Bus.
development machine	Machine that combines a client to a database server and a Java development environment such as Visual Cafe Database Edition.
DML	Data Modification Language, a subset of SQL.
Dynamic List Binding	A property that allows you to bind and display multiple records from a single column in a list or combobox component. Also referred to as list binding.

E

exception	An event that occurs during the execution of your program that interferes with, disrupts, or stops the normal flow of instructions.

F

filter	A way to process data that allows you to screen out items you do not want to view or access, but do not want to remove from the database.
foreign key	Column(s) in database table whose values match the primary key in another table. Foreign key columns may or may not be defined as Unique or NOT NULL.

I

invisible component

Component that is not visible at run time, such as a Timer. It does not extend from the Java Component class. In the Form Designer, an invisible component is represented by an icon that does not affect the form layout.

J

JDBC API

An API developed by JavaSoft as a standard SQL-level database protocol. JDBC, based on ODBC, is a database extension to Java. The Visual Cafe Database Development Edition implementation of the JDBC API includes the Symantec JDBC components, in addition to the base JDBC. While it is not an acronym, JDBC is often thought of as the Java Database Connectivity protocol.

JDBC-ODBC bridge

The bridge between Java code and ODBC. It supports an ODBC subprotocol of the JDBC API. The JDBC API requires a URL to establish a connection to a database; the subprotocol is part of this URL syntax. For example:

jdbc:odbc://*subname*

When attempting to connect to a JDBC driver, the Java program makes a connect call, passing the URL to the driver. If the driver recognizes the information in the URL, including the subprotocol, the connection can be made.

JdbcConnection

A bean, used with the JDBC API, that represents a connection to a database. It has properties for establishing connection parameters in accordance with JDBC. Dragging a datasource item from dbNAVIGATOR to a project creates ConnectionManager and JdbcConnection objects, which manage connections to a datasource. You have one ConnectionManager object per project, and one JdbcConnection object per datasource. You can drag either of these objects to the Component Library.

join	Any database query that produces data from more than one table. As its name suggests, a join brings together data from the specified tables.L

L

list binding	See Dynamic List Binding.

M

master-detail relationship	A one to many relationship between two database tables. The join property of the detail `Query Navigator` component defines the column of the master view which is the common attribute in the one to many relationship between the tables.
mediator	Bean that lets you make a component databound. It is a bridge between the component and the `Query Navigator` component via the DataBus.

O

operator	Acharacter that represents a type of operation such as addition, multiplication or division. May also represent boolean search terms, such as `AND`, `OR`, or `NOT`.
ODBC	Open DataBase Connectivity, a standard interface for communicating with databases. The dbANYWHERE server uses this interface to communicate with many of the databases it supports. The JDBC-ODBC Bridge also uses this interface.
optimistic concurrency	Locking technique that maximizes concurrency. The database management system (DBMS) locks data that is being modified but leaves data that is being viewed.

P

ping: Packet INternet Groper	Network message that a computer transmits to check for the presence, connectedness, and responsiveness

of another computer. The other computer responds by echoing the message back to the first computer.

primary key — Column(s) in a database table that uniquely identify a record. A value in a primary key column cannot be NULL and must be unique.

projection binding — Data binding to one row in a database column.

Q

QueryNavigator — A bean, used with the JDBC API, that manages a set of records. Dragging a Data Table item from a dbNAVIGATOR to a form in a project creates a top-level `Record Definition` object and a `Query Navigator` object, which is contained by the form. You can think of the `Record Definition` component as the data, and the `Query Navigator` component as a way of looking at the data.

R

RecordDefinition — A bean, used with the JDBC API to define and access a row of data in a database table. Dragging a Data Table item from dbNAVIGATOR to a form in a project creates a top-level Record Definition object and a `Query Navigator` object, which is contained by the form.

refresh — Updates the dbNAVIGATOR window to reflect changes made to database tables and columns it is displaying.

relational database — A database based on the relational model created by E.F. Codd. The database is comprised of tables, which contain columns and rows (or fields and records). Relational databases allow records in different tables to be linked if they share the same value in one field of each table.

RelationViewPlus — Databound component that shows a view of a database and is used with the dbANYWHERE API. It provides a simple, property-driven solution for defining and maintaining a result set. It is invisible at

	run time but contains methods used to handle navigation and data manipulation operations on the data in the result set. (The “Plus” was added when the data binding changed in later versions of Visual Cafe Database Edition.)
result set	Set of data which is returned by a query on a database(s). An example of a result set might be the data from all the rows in an employee database table which have 200 as their department identification number. `Stored Procedure Adapters` can use result sets as input.
run time	Time when your applet or application is running.

S

Select Clause	A SQL statement using SELECT. See SQL Statement.
Session	For the dbANYWHERE API, when you use the database wizard or drag a dbANYWHERE Server item to a project, Visual Cafe Database Edition creates a Session object, indicating the dbANYWHERE Server you are connected to through sockets.
server	Database server: computer that stores and manages a database. dbANYWHERE Server: computer that runs dbANYWHERE, the middleware database software from Symantec which supports three-tier architecture for database access. File server: computer that stores programs and data files. Web server: computer that stores and sends out Web pages in response to HTTP requests from Web browsers.
SQL	Standard Query Language which is used for retrieving information from a relational database.
SQL statement	A SQL string used to query a database and thereby define a view. If the string is in the standard SELECT

statement format, the Visual Cafe Databound wizards and Join Definition Editor can parse the string and edit it for you. If the string is not in the standard SELECT statement format, you need to use the SQL Statement dialog box to edit the string manually.

Standard format

SELECT *column*[, *column* ...] FROM *table* WHERE *condition*

Example

SELECT emp_fname, emp_lname, location_id FROM emp_info WHERE location_id BETWEEN 10 AND 20

stored procedure

Collection of SQL statements that is named and precompiled.

Procedures and triggers store procedural SQL statements in a database for use by all applications.

Procedures are invoked with a CALL_statement, and use parameters to accept values and return values to the calling environment. Procedures can also return result sets to the caller. Procedures can call other procedures and fire triggers.

Triggers are associated with specific database tables. They are invoked automatically (fired) whenever rows of the associated table are inserted, updated or deleted. Triggers do not have parameters and cannot be invoked by a CALL statement. Triggers can call procedures and fire other triggers.

User-defined functions are one kind of stored procedure that returns a single value to the calling environment. User-defined functions do not modify parameters passed to them. They broaden the scope of functions available to queries and other SQL statements.

Stored Procedure Adapter

A bean, allows you to access a stored procedure from a database and use the result set in a project.

T

table

A collection of records.

top-level component

A container component that can appear only at the top level in the Objects view of the Project window. Also called a form. Applet, Frame, Window, and some dialog components are top-level components. In the Component Library, the Visual Cafe top-level components are all in the Forms group. When you open a top-level component in the Form Designer, the component boundaries are the bounds of the window. You can position other components, such as text, buttons, and graphics, on the top-level component in the Form Designer; these components are contained by the top-level component.

V

validation

Process of verifying the integrity of data within the paramters of a container. For example, you can implement validation that requires data to fall within specified ranges of values.

view

Virtual database table that exists only in memory. A `Query Navigator` and a `Record Definition` component represents a view. A SQL statement defines the view.

- Master view: View that is used as the basis for a detail view.
- Detail view: View that is related to a master view. A detail view's join defines a master-detail relationship.
- A view can be a master view or a detail view or both or neither.

I N D E X

D

E

F

G

I