

## taskspro.doc

- *Contents*: This file contains the procedure topics for Visual Cafe Database Development Edition (dbDE), and some overview topics
- *Dialog box topics*: A dialog box topic can be accessed only from a dialog box or a Details button in a procedure topic (not from the index, no keywords) – because a dialog box topic doesn't make sense unless you are looking at the dialog box.

### History

2/12/97: Created and added to project

9/2/97: Updated file to reflect new product name Visual Cafe Database Development Edition.

11/97: Rewrote for JDBC subprotocols ODBC and dbAW. Also corrected some display and naming problems, and added browse sequences.

## Topics, in browse (+) order

## Intro



## Welcome to Visual Cafe Database Development Edition

{button How

To,AL('Creating\_a\_dbAWARE\_project;Connecting\_to\_a\_dbANYWHERE\_Server;Adding\_a\_table\_to\_your\_project;Creating\_a\_data\_source\_table\_from\_a\_template;Adding\_a\_dbAWARE\_field\_to\_an\_existing\_form;Specifying\_SQL\_SELECT\_statements;Specifying\_JDBC\_database\_connection\_URL',0,',')}) {button See  
Also,AL('dbNAVIGATOR\_window;Visual\_Cafe\_components\_and\_containers;Wizards\_using;Mediator\_overview;Creating\_a\_MultiView;Database\_Development  
Edition\_Extension\_Overview;Grid\_component;RelationView\_component;Session\_component;ConnectionInfo\_component;ConnectionManager\_component;RecordDefinition\_component;JdbcConnection\_component;QueryNavigator\_component;Customizing\_Database\_Development\_Edition\_code',0,',')})

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

### The environment

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

- [Visual Cafe](#)
- [Database Development Edition Extension](#)
- [dbANYWHERE Server](#)

You can assemble complete Java applets and applications from a library of standard and third-party objects without writing a lot of Java code. These Java programs can be database-aware. Visual Cafe Database Development Edition seamlessly integrates visual and source development of Java software, letting you switch effortlessly between visual and source views. Modifications that you make in the visual view are immediately reflected in the source code. And the source code automatically updates changes to the visual view.

### This help system

The Visual Cafe Database Development Edition help system is made of the Visual Cafe online help (vcfe.hlp), help on the Database Development Edition extensions (vcafepro.hlp), and the dbANYWHERE online help (dbaw.hlp). Note that if the Visual Cafe online help says that a feature is in the Professional edition, that means it is also in the Database edition.



## What's new in version 2.5 of the Database Development Edition

{button How

To,AL('Applets\_step\_overview;Applications\_step\_overview;Applications\_native\_step\_overview;Applications\_native\_console\_step\_overview;DLLs\_native\_step\_overview;Creating\_beans\_overview',0,'','')) {button See

Also,AL('Overview\_of\_Visual\_Caf;Working\_Visually;Visual\_Caf\_as\_a\_Development\_Environment;The\_Differences\_between\_Applets\_and\_Applications;GUI\_development\_with\_Visual\_Caf;Visual\_Cafe\_Tools\_overview;How\_is\_my\_Work\_Kept\_In\_Sync\_overview;How\_WYSIWYG\_is\_Visual\_Cafe\_overview;Workspace\_customization\_overview;Creating\_native\_overview;Native\_libraries\_and\_DLLs\_overview',0,'',''))

You can learn about the [master/detail](#) feature added to the [JDBC API](#) by looking at these new or revised topics:

- [Adding database functionality to a project](#)
- [Creating a view and a master/detail relationship](#)
- [Changing a view](#)
- [Deleting a view](#)
- [Understanding JDBC master/detail capabilities](#)

In addition, these new topics were added in this version of the help:

- [Managing QueryNavigator components in a Java program](#)
- [Closing a form when its QueryNavigator closes](#)

See your What's New documentation for more complete information.



## Visual Cafe Database Development Edition overview

{button How

To,AL('Creating\_a\_dbAWARE\_project;Connecting\_to\_a\_dbANYWHERE\_Server;Adding\_a\_table\_to\_your\_project;Creating\_a\_data\_source\_table\_from\_a\_template;database\_environment\_options\_db;Adding\_a\_dbAWARE\_field\_to\_an\_existing\_form;Specifying\_SQL\_SELECT\_statements;Specifying\_JDBC\_database\_connection\_URL','0','')' } {button See Also,AL('dbNAVIGATOR\_window;Visual\_Cafe\_components\_and\_containers;Wizards\_using;Mediator\_overview;Creating\_a\_MultiView;Grid\_component;RelationView\_component;Session\_component;ConnectionInfo\_component;ConnectionManager\_component;RecordDefinition\_component;JdbcConnection\_component;QueryNavigator\_component;Customizing\_Database\_Development\_Edition\_code','0','')' }

Visual Cafe Database Development Edition includes the same features as Visual Cafe Professional Development Edition, plus provides features that allow your application to communicate with a database. Visual Cafe does this by generating code compliant with the [JDBC API](#) and a JDBC subprotocol, such as

- [dbAW](#), supported by the Symantec [dbANYWHERE](#) middleware application included with Visual Cafe Database Development Edition,
- [ODBC](#), supported by the [JDBC-ODBC bridge](#), or
- another subprotocol.

Alternatively, you can use the [dbANYWHERE API](#), which was used in Visual Cafe Database Development Edition before version 2.1 and in Visual Cafe Pro, and is supported by the dbANYWHERE middleware application.

The three main features of the Database Development Edition are:

- database wizards
- [dbNAVIGATOR](#) window
- database-aware [components](#)

### Database wizards

The Visual Cafe Database Development Edition wizards step you through complicated processes, such as

- creating a new [form](#) over a data source table
- adding data source table connectivity to an existing form
- defining [master/detail joins](#)
- creating tables and forms from predefined templates

The wizards can create code compliant with either the JDBC API or the dbANYWHERE API.

### dbNAVIGATOR

The dbNAVIGATOR provides a hierarchical view of data source cataloging information known as meta-data. This view lets you see dbANYWHERE Servers, available data sources, and the tables and columns in each data source.

### Database components

Visual Cafe Database Development Edition includes these database components:

- JDBC components, for use with the JDBC API,
- dbANYWHERE components, for use with the dbANYWHERE API, and
- [dbAWARE](#) components, for use with either API.

These components encompass and simplify database connectivity and facilitate binding to database columns. Database-aware forms communicate with data sources through the dbANYWHERE Server or another [ODBC](#) server. The [data binding](#) is the same whether your data is being obtained through the JDBC API or the dbANYWHERE API.

### dbANYWHERE Server

The dbANYWHERE Server is a tool that manages connections between a Java program and one or more databases. It allows you to embed database access code in your Java programs; this code is not dependent on any particular database engine. The dbANYWHERE Server provides a 100% pure JDBC solution as well as a powerful dbANYWHERE API. The code that Visual Cafe generates for Java applets and applications handles the binding of data between visual components and the data access components.



## Using the JDBC API

```
{button How
To,AL('Creating_a_dbAWARE_project;Connecting_to_a_dbANYWHERE_Server;Adding_a_table_to_your_project;Creating_a_d
ata_source_table_from_a_template;database_environment_options_db;Adding_a_dbAWARE_field_to_an_existing_form;Specify
ing_SQL_SELECT_statements;Specifying_JDBC_database_connection_URL;Closing_form_when_QueryNavigator_closes_how
to',0,',',',')} {button See
Also,AL('dbNAVIGATOR_window;Visual_Cafe_components_and_containers;Wizards_using;Mediator_overview;Database
Development
Edition_Extension_Overview;ConnectionManager_component;RecordDefinition_component;JdbcConnection_component;Query
Navigator_component;Data_type_mapping_overview;Customizing_Database_Development_Edition_code;Master_detail_JDBC_
overview;QueryNavigator_close_overview',0,',',',')}
```

Visual Cafe Database Development Edition can generate code compliant with the [JDBC API](#). Alternatively, you can use the [dbANYWHERE API](#), which was used in Visual Cafe Database Development Edition before version 2.1 and in Visual Cafe Pro. You specify which API you want to use and the default [component](#) per data type by choosing Tools ► Environment Options

► Database tab.

### What is JDBC?

Visual Cafe generates JDBC code to connect you to a database. The Database Development 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. 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 database-aware applets might be a corporate phone list or an email form which allows you to receive feedback from a Web page. An application might be a stock reporter, which gathers information from a database on the Web and stores it in a database on your local machine.

### Using database components

Visual Cafe Database Development Edition includes a group of JDBC components, for use with the JDBC API, and a group of [dbAWARE](#) components, which are visible components for use with either API. The [dbANYWHERE](#) group includes invisible components for use with the [dbANYWHERE](#) API. The [data binding](#) is the same whether your data is being obtained through the JDBC API or the [dbANYWHERE](#) API.

Following are the JDBC components:

- [ConnectionManager](#)
- [JdbcConnection](#)
- [RecordDefinition](#)
- [QueryNavigator](#)

### Specifying JDBC subprotocols

When using the JDBC API, you specify a subprotocol, either [dbAW](#) (supported by the Symantec [dbANYWHERE](#) middleware application) or [ODBC](#) (supported by the [JDBC-ODBC bridge](#)). You can also specify another third-party subprotocol in the URL.

### Using the dbANYWHERE Server

The Symantec [dbANYWHERE](#) Server supports the JDBC API ([dbAW](#) subprotocol) and the [dbANYWHERE](#) API. The [dbANYWHERE](#) Server is a tool that manages connections between a Java program and one or more databases. It allows you to embed database access code in your Java programs; this code is not dependent on any particular database engine. The code that Visual Cafe generates for Java applets and applications handles the binding of data between visual components and the

data access components.

#### **Considerations when using the ODBC subprotocol or another subprotocol**

dbANYWHERE must be running at design time in order for you to be able to use the database wizards and dbNAVIGATOR. The code that is automatically generated by the wizards and when dragging from dbNAVIGATOR to a project only works with the dbAW subprotocol. You need to change the URL in the [JdbcConnection](#) component to support another subprotocol.

#### **Working with both APIs**

Your Environment Options settings in the Database view apply when you are running a database wizard or dragging from dbNAVIGATOR to a project. You can work on one project using the JDBC API and another using the dbANYWHERE API simultaneously, but you need to choose the correct API when you are using a database wizard or dragging from dbNAVIGATOR to a project.



## Using the dbANYWHERE API

{button How

To,AL('Creating\_a\_dbAWARE\_project;Connecting\_to\_a\_dbANYWHERE\_Server;Adding\_a\_table\_to\_your\_project;Creating\_a\_data\_source\_table\_from\_a\_template;database\_environment\_options\_db;Adding\_a\_dbAWARE\_field\_to\_an\_existing\_form;Specifying\_SQL\_SELECT\_statements',0,'','')) {button See

Also,AL('dbNAVIGATOR\_window;Visual\_Cafe\_components\_and\_containers;Wizards\_using;Mediator\_overview;Creating\_a\_MultiView;Database\_Development

Edition\_Extension\_Overview;RelationView\_component;Session\_component;ConnectionInfo\_component;Customizing\_Database\_Development\_Edition\_code',0,'',''))}

Visual Cafe Database Development Edition includes the Visual Cafe Professional Development Edition features, special data source connectivity [components](#) and wizards, and the [dbANYWHERE Server](#). When you are using the database wizards and dragging from dbNAVIGATOR, Visual Cafe can automatically generate Java code that includes calls to the [dbANYWHERE API](#). Alternatively, Visual Cafe can generate code using the [JDBC API](#), which is also supported by dbANYWHERE Server. You specify which API you want to use and the default [component](#) per data type by choosing Tools ► Environment Options

► Database tab.

Included in Visual Cafe Database Development Edition are dbANYWHERE database [components](#), for use with the dbANYWHERE API, and [dbAWARE](#) components, which are visible components for use with either API. There are also JDBC database components, for use with the JDBC API. The [data binding](#) is the same whether your data is being obtained from the dbANYWHERE API or the JDBC API.

The proprietary dbANYWHERE API:

- supports simultaneous access to multiple databases, both local and remote
- provides simple database functionality in powerful, easy-to-use classes
- supports the three-tier system architecture, where the first tier is the client program, the second tier is the dbANYWHERE middleware and database drivers, and the third tier is made of the database servers

Use the dbANYWHERE API to:

- Manage transactions, database connections, data buffering, heterogeneous joins, and all other database communications.
- Simultaneously access multiple databases.
- Manage database sharing capabilities. (This lets you significantly reduce the number of database licenses.)

Following are the dbANYWHERE database access components:

- [Session](#)
- [ConnectionInfo](#)
- [RelationViewPlus](#)

### Note

- Your Environment Options settings in the Database view (specifying both the API and the default component per data type) apply when you are running a database wizard or dragging from dbNAVIGATOR to a project.
- You can work on one project using JDBC API and another using dbANYWHERE API simultaneously, but you need to choose the correct API when you are using a database wizard or dragging from dbNAVIGATOR to a project.





## Using the dbANYWHERE application

{button How

To,AL('Creating\_a\_dbAWARE\_project;Connecting\_to\_a\_dbANYWHERE\_Server;Adding\_a\_table\_to\_your\_project;Creating\_a\_data\_source\_table\_from\_a\_template;Adding\_a\_dbAWARE\_field\_to\_an\_existing\_form;Specifying\_SQL\_SELECT\_statements;Specifying\_JDBC\_database\_connection\_URL',0,'') } {button See  
Also,AL('dbNAVIGATOR\_window;Visual\_Cafe\_components\_and\_containers;Wizards\_using;Mediator\_overview;Creating\_a\_MultiView;Database\_Development\_Edition\_Extension\_Overview',0,'') }

The Visual Cafe Database Development Edition provides the user interface to design forms that interact with a data source. If you choose, you can use [dbANYWHERE](#) as the software element that defines and manages the data source connectivity.

Many connectivity tasks are documented in the dbANYWHERE online help.

You can access the dbANYWHERE information from the

- Visual Cafe Database Development Edition online help Contents list
- Visual Cafe Database Development Edition Help menu, dbANYWHERE topic
- dbANYWHERE Help menu



## Adding database functionality to a project

```
{button
Concepts,AL('dbNAVIGATOR_window;Visual_Cafe_components_and_containers;Wizards_using;Mediator_overview;Creating_a
_MultiView;Database Development
Edition_Extension_Overview;Using_the_JDBC_API;Using_the_dbANYWHERE_API;Customizing_Database_Development_Edit
on_code;Master_detail_JDBC_overview;QueryNavigator_close_overview',0,'')} {button See
Also,AL('Grid_component;RelationView_component;Session_component;ConnectionInfo_component;ConnectionManager_com
ponent;RecordDefinition_component;JdbcConnection_component;QueryNavigator_component;Specifying_SQL_SELECT_state
ments;Specifying_JDBC_database_connection_URL',0,'')}
```

Visual Cafe Database Development Edition provides many tools to help you quickly add database access to your Java programs. This includes database wizards, dbNAVIGATOR, database-aware components, and [mediators](#).

Before you add any database-aware [components](#) to your project or use a database wizard, make sure you have chosen the API you want to use, [JDBC API](#) or [dbANYWHERE API](#), in your Environment Options. You can also map default components to data types. See [Setting the database environment options](#) for more information.

### Note

- You can simultaneously work on one project using the JDBC API and another using the dbANYWHERE API, but you need to choose the correct API when you are using a database wizard or dragging from dbNAVIGATOR to a project. The [data binding](#) is the same for both APIs.
- dbANYWHERE must be running at design time in order for you to be able to use the database wizards and dbNAVIGATOR. For the JDBC API, the code that is automatically generated by the wizards and when dragging from dbNAVIGATOR to a project defaults to the dbAW subprotocol. You need to change the URL in the [JdbcConnection](#) component to support another subprotocol.

### To create a new database-aware project

You can use the dbAWARE Project wizard to quickly create a new database-aware project:

1. Choose File ► New Project.
2. Select the dbAWARE Project Wizard, and click OK.
3. Complete the wizard.

Press F1 in any wizard page to receive help on that page.

### To add a database table to your project

The Add Table wizard helps you add a table to your project. If your project already has the components that represent a [view](#), you can create a [master/detail relationship](#). For the [JDBC API](#), the [QueryNavigator](#) and [RecordDefinition](#) components together represent a view. For the [dbANYWHERE API](#), a [RelationViewPlus](#) component represents a view. If you are using the JDBC API, you can create cross-form [joins](#).

1. Select a [container](#) in the project, then choose Insert ► Add Table Wizard.
2. Complete the wizard.

Press F1 in any wizard page to receive help on that page.

### To add a database-aware field from the dbNAVIGATOR

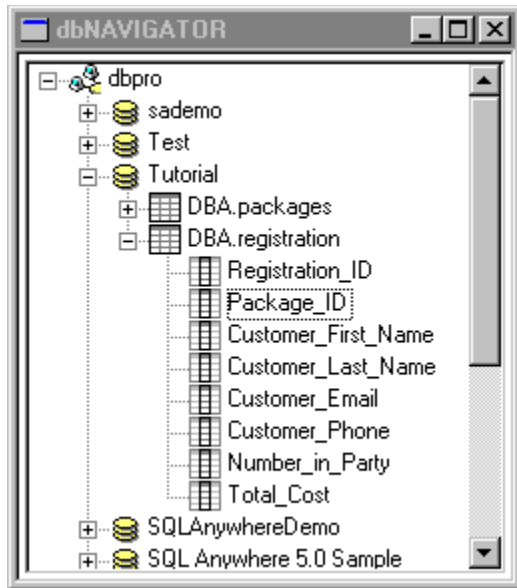
1. In dbNAVIGATOR, connect to a data source.



[Details on this step](#)

3. Expand a Database Table item (click the +).

Following is an example of an expanded Database Table.



4. Drag a Database Column item to the Form Designer or Project window.

Visual Cafe Database Development Edition adds a database-aware component to the [form](#) or [container](#) you dragged it to. The component uses the column name as the Text property. Any supporting database-aware components are automatically added, if they are not already present. For the JDBC API, this includes the [ConnectionManager](#), [JdbcConnection](#), [RecordDefinition](#), and [QueryNavigator](#) components. For the dbANYWHERE API, this includes the [Session](#), [ConnectionInfo](#), and [RelationViewPlus](#) components.

5. Position and size the component as needed.

#### To add a database-aware field from the Component Library or Palette

To use database-aware visible components, you need to have supporting database-aware invisible components in your project. You can add them before or after adding database-aware fields. For the JDBC API, this includes the [ConnectionManager](#), [JdbcConnection](#), [RecordDefinition](#), and [QueryNavigator](#) components. For the dbANYWHERE API, this includes the [Session](#), [ConnectionInfo](#), and [RelationViewPlus](#) components. If you use a database wizard or drag from dbNAVIGATOR, these supporting components are added automatically for you.

1. Drag a database-aware component from the Component Library or Palette to your form or project.

Visual Cafe Database Development Edition adds a database-aware component to the [form](#) or [container](#) you dragged it to.

2. In the Property List, set the Data Binding property, which includes [QueryNavigator](#) alias name and field name, or [RelationViewPlus](#) object name and field name.

**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 QueryNavigator you enter does not have to be contained by the same form as the component.


#### To use the same data source across forms in a project


- Set the Data Binding property of a database-aware component to use a [QueryNavigator](#) alias name and field name, or [RelationViewPlus](#) object name and field name.

**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 QueryNavigator you enter does not have to be contained by the same form as the component.

#### To make a non-database-aware component database-aware

- You can use a [mediator](#) to make a component database-aware.

 [Details on adding a mediator to a project](#)

 [Details on adding a mediator to component code](#)



## Specifying SQL SELECT statements

```
{button  
Concepts,AL('dbNAVIGATOR_window;Visual_Cafe_components_and_containers;Wizards_using;Mediator_overview;Creating_a  
_MultiView;Database Development  
Edition_Extension_Overview;Using_the_JDBC_API;Using_the_dbANYWHERE_API;Customizing_Database_Development_Edit  
on_code;QueryNavigator_close_overview',0,'')} {button See  
Also,AL('Grid_component;RelationView_component;Session_component;ConnectionInfo_component;ConnectionManager_com  
ponent;RecordDefinition_component;JdbcConnection_component;QueryNavigator_component;Specifying_JDBC_database_con  
nection_URL;Adding_a_dbAWARE_field_to_an_existing_form',0,'')}
```

Visual Cafe Database Development 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 pieces, some of which are optional:

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

For the [JDBC API](#):

- *columns* and *table* are specified in the Table Name property of the [RecordDefinition](#) component.
- *filter* is specified in the Filter property of the [QueryNavigator](#) component.
- *sort-order* is specified in the Sort Order property of the QueryNavigator component.

For the [dbANYWHERE API](#), the entire SELECT statement is specified in the [RelationViewPlus](#) Select Clause property.

Remember that you can press F1 after selecting a property or component or while in a dialog box to get help.



## Specifying a JDBC database connection URL

```
{button  
Concepts,AL('dbNAVIGATOR_window;Visual_Cafe_components_and_containers;Wizards_using;Mediator_overview;Creating_a  
_MultiView;Database Development Edition_Extension_Overview;Using_the_JDBC_API',0,'','')} {button See  
Also,AL('ConnectionManager_component;RecordDefinition_component;JdbcConnection_component;QueryNavigator_compone  
nt;Adding_a_dbAWARE_field_to_an_existing_form',0,'','')}
```

Visual Cafe Database Development Edition lets you specify a URL for connecting to a database through the [JDBC API](#). The JDBC API requires a URL to establish a connection to a database; the subprotocol is part of this URL syntax. 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.

You specify the URL in the URL property of the [JdbcConnection](#) component.

### dbAW subprotocol

This JDBC subprotocol is supported by Symantec [dbANYWHERE](#), which is a JDBC driver. Here is the URL syntax:

**jdbc:dbaw://host-name-or-IP-address:port-number/data-source-name**

### ODBC or another subprotocol

The ODBC subprotocol is supported by the [JDBC-ODBC bridge](#), which is a free JDBC driver. (Each JDBC driver supports certain subprotocols.) Here is the URL syntax:

**jdbc:odbc://subname**

You can specify another subprotocol in place of **odbc** in this URL.



## Opening projects and files developed in previous versions

{button

Concepts,AL('dbNAVIGATOR\_window;Visual\_Cafe\_components\_and\_containers;Wizards\_using;Mediator\_overview;Creating\_a  
\_MultiView;Database Development  
Edition\_Extension\_Overview;Using\_the\_JDBC\_API;Using\_the\_dbANYWHERE\_API;Customizing\_Database\_Development\_Edit  
on\_code',0,'') {button See

Also,AL('Creating\_a\_dbAWARE\_project;Connecting\_to\_a\_dbANYWHERE\_Server;Adding\_a\_table\_to\_your\_project;Creating\_a  
\_data\_source\_table\_from\_a\_template',0,'')

Projects and files developed in Visual Cafe Database Development Edition before version 2.1 or in Visual Cafe Pro use the [dbANYWHERE API](#) and the previous [data binding](#) method. Any new database-aware [components](#) you add will use the new data binding method, but existing database-aware components will use the previous data binding method.

**Note** You need to set your Environment Options to use the dbANYWHERE API if you want to continue using this API. You cannot automatically convert existing files that use the dbANYWHERE API to the [JDBC API](#).

Data Types, supported



## JDBC-to-native data type mapping

```
{button How To,AL('Creating_a_dbAWARE_project;Projects_overview;Form_Designer_F1;Editing_a_form_howto',0,'')}
{button See
Also,AL('dbNAVIGATOR_window;Visual_Cafe_components_and_containers;Wizards_using;Mediator_overview;Creating_a_MultiView;Database_Development
Edition_Extension_Overview;Using_the_JDBC_API;Customizing_Database_Development_Edition_code',0,'')}
```

Following are some mappings between data types in databases and JDBC.

**Note** Refer to the JDBC specification for the mapping between SQL types and Java types.

### Informix

Native 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	Behaves the same way as	FLOAT
PRECISION	FLOAT	
FLOAT(n)	Stores double-precision floating-point numbers corresponding to the double data type in C	FLOAT
INT	A synonym for INTEGER	INTEGER
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	SMALLINT



TEXT	-32,767 to +32,767	
VARCHAR(m,r)	Stores any kind of text data	LONGVARCHAR
	Stores multibyte strings of letters, numbers, and symbols of varying length; collation is code-set dependent	VARCHAR

#### Oracle

Native 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

Native 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

Native 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**

<b>Native data type</b>	<b>JDBC mapping</b>
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	FLOAT, REAL
(single precision floating point)	
double	DOUBLE
(double precision floating point)	
date	DATE
timestamp	TIMESTAMP
time	TIME
binary ( size )	BINARY
long binary	LONGVARBINARY

#### **Microsoft Access**

<b>Native data type</b>	<b>JDBC mapping</b>
Bit	BIT
Byte	TINYINT
Binary	BINARY
Counter	INTEGER
(auto-increment)	
Currency	DECIMAL
Datetime	TIMESTAMP
Single	FLOAT
(single precision floating point)	
Double	DOUBLE
(double precision floating point)	
Short	SMALLINT
Long	INTEGER
Long Text	LONGVARCHAR
Long Binary	LONGVARBINARY
Text	CHAR

dbNAVIGATOR



## Understanding the dbNAVIGATOR window

{button How

To,AL('dbNAVIGATOR\_pop\_up\_menu;Connecting\_to\_a\_dbANYWHERE\_Server;Connecting\_to\_a\_database;Using\_more\_than\_one\_Data\_Source;Disconnecting\_a\_database;Disconnecting\_from\_a\_dbANYWHERE\_Server;Refreshing\_dbNAVIGATOR;Creating\_a\_data\_source',0,';')) {button See

Also,AL('Grid\_component;RelationView\_component;Session\_component;ConnectionInfo\_component;ConnectionManager\_component;RecordDefinition\_component;JdbcConnection\_component;QueryNavigator\_component;Data\_type\_mapping\_overview',0,';'))}

After connecting to a dbANYWHERE Server, you can use the window called the dbNAVIGATOR as an easy way to view a list of dbANYWHERE servers and [data sources](#) that are currently connected. To access dbNAVIGATOR, choose View (or Window) ► dbNAVIGATOR.

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

Item	Description
dbANYWHERE Server items	<p>Represents a dbANYWHERE Server that can be accessed by the computer that Visual Cafe Database Development Edition is running on.</p> <p>For the <a href="#">JDBC API</a>, you cannot drag a dbANYWHERE Server item to a form in a project.</p> <p>For the <a href="#">dbANYWHERE API</a>, when you drag a dbANYWHERE Server item to a form it becomes a <a href="#">Session</a> object, indicating the dbANYWHERE Server you are connected to through sockets.</p>
Data Source items	<p>Represents a database that is connected to the associated dbANYWHERE Server.</p> <p>For the JDBC API, dragging a Data Source item to a form in a project creates <a href="#">ConnectionManager</a> and <a href="#">JdbcConnection</a> objects, which manage connections to a data source. You have one ConnectionManager object per project, and one JdbcConnection object per data source.</p> <p>For the dbANYWHERE API, when you drag a Data Source item to a project it becomes a <a href="#">ConnectionInfo</a> object, showing the database you are connected to.</p>
Database Table items	<p>Represents a table in the associated database. When you connect to a database, dbNAVIGATOR displays the tables and columns that are in the database.</p> <p>For the JDBC API, dragging a Database Table item to a form in a project creates a top-level <a href="#">RecordDefinition</a> object and a <a href="#">QueryNavigator</a> object, which is contained by the form.</p> <p>For the dbANYWHERE API, when you drag a Database Table item to a form it becomes a <a href="#">RelationViewPlus</a> object, showing a <a href="#">view</a> of the database.</p>

Database Column items	Represents a column in the associated database table. Dragging a Database Column item to a project creates a <a href="#">database-aware component</a> . You can specify the default component type by choosing Tools ► Environment Options ► Database tab.
--------------------------	--

When you drag to a project an item lower in the hierarchy, the items you need higher in the hierarchy are added to the project for you.

Within the dbNAVIGATOR window, you can expand and contract the items in the hierarchy. If you expand a dbANYWHERE Server item, you see the Data Source items on the server. If you expand a Data Source item, you see the Database Table items in the data source. If you expand a Database Table item, you see the Database Column items in the table.

There are many database manipulation functions you can carry out by using the dbNAVIGATOR window. Some of these tasks include:

[Connecting to a dbANYWHERE Server](#)

[Connecting to a data source](#)

[Adding a table to your project](#)

[Refreshing dbNAVIGATOR](#)

[Disconnecting from a data source](#)

[Disconnecting from a dbANYWHERE Server](#)



## Using the dbNAVIGATOR pop-up menu

```
{button Concepts,AL('dbNAVIGATOR_window',0,'')} {button See  
Also,AL('Connecting_to_a_dbANYWHERE_Server;Connecting_to_a_database;Using_more_than_one_Data_Source;Disconnec  
ting_a_database;Disconnecting_from_a_dbANYWHERE_Server;Refreshing_dbNAVIGATOR;Creating_a_data_source;dbANYW  
HERE_Server_dialog_box;Logon_dialog_box',0,'')}
```

When you right-click in the dbNAVIGATOR window, a pop-up menu displays with the following commands:

### **Insert Server**

Opens the dbANYWHERE Server dialog box that lets you connect to a dbANYWHERE Server. This command is available only if a dbANYWHERE Server is selected.

### **Delete Server**

Deletes the selected dbANYWHERE Server from the dbNAVIGATOR. This command is available only if a dbANYWHERE Server is selected.

### **Connect**

Opens the Logon dialog box that lets you logon to a [data source](#). This command is available only if a Data Source is selected.

### **Disconnect**

Disconnects the selected database.

### **Refresh**

Refreshes the selected database's dbNAVIGATOR tables and columns information. This command is useful for multiple concurrent users. This command is available only if a dbANYWHERE Server or Data Source is selected.



## Connecting to a dbANYWHERE Server

```
{button Concepts,AL('dbNAVIGATOR_window',0,'')} {button See  
Also,AL('dbNAVIGATOR_pop_up_menu;Connecting_to_a_database;Using_more_than_one_Data_Source;Disconnecting_a_dat  
abase;Disconnecting_from_a_dbANYWHERE_Server;Refreshing_dbNAVIGATOR;Creating_a_data_source',0,'')}
```

To access a data source, you must first connect to a dbANYWHERE Server that has the data source registration.

1. Choose Window ► dbNAVIGATOR.
2. If the dbNAVIGATOR window appears, select a dbANYWHERE Server, then right-click and choose Insert Server.
3. Use the dialog box to connect to a dbANYWHERE Server.



[Details on this step](#)



## Connecting to a data source

```
{button Concepts,AL('dbNAVIGATOR_window',0,'')} {button See  
Also,AL('dbNAVIGATOR_pop_up_menu;Connecting_to_a_dbANYWHERE_Server;Using_more_than_one_Data_Source;Discon  
necting_a_database;Disconnecting_from_a_dbANYWHERE_Server;Refreshing_dbNAVIGATOR;Creating_a_data_source;wizar  
ds_using',0,'')}
```

A data source is what identifies a database to a database application. When you set a data source for your project, you are really packaging the information that enables your application to connect to the database you want to access. The data source includes details which are needed for connection, like the name of the database, its server, and its network location.

Once you have established a dbANYWHERE server connection, you can select a data source from the list of registered data sources.

**Note** You can connect to the data source at any time. When developing a Java applet that uses a database, Visual Cafe Database Development Edition maintains one persistent connection. A separate connection is made automatically when running the applet, which is disconnected once the applet is terminated.

1. Choose Window ► dbNAVIGATOR.

If you display the dbNAVIGATOR and it is empty, the dbANYWHERE Server dialog box displays.

► Details on the dbANYWHERE Server dialog box

2. Expand the dbANYWHERE server item (click the +).

The [Data Sources](#) on the server appear.

3. Expand a Data Source item.

The Logon dialog box appears.

If you do not see the data source you want, make sure that you have selected and connected to the correct dbANYWHERE server and that you have registered the data source.

► Details on connecting to a dbANYWHERE Server

4. Type the user name and password for the data source, then click OK.

► Details on this step



## Using more than one data source

{button See  
Also,AL('dbNAVIGATOR\_pop\_up\_menu;Connecting\_to\_a\_dbANYWHERE\_Server;Connecting\_to\_a\_database;Disconnecting\_a\_database;Disconnecting\_from\_a\_dbANYWHERE\_Server;Refreshing\_dbNAVIGATOR;Creating\_a\_data\_source;RelationView\_component;Session\_component;ConnectionInfo\_component;ConnectionManager\_component;JdbcConnection\_component;RecordDefinition\_component;QueryNavigator\_component',0,'')}

It is common in medium- and large-scale database projects to use data from more than one data source and/or from more than one server.

**Note** When you drag from dbNAVIGATOR to a project an item lower in the hierarchy, the items you need higher in the hierarchy are added to the project for you.

### JDBC API

- [ConnectionManager](#) and [JdbcConnection](#) objects manage connections to a data source. You have one ConnectionManager object per project, and one JdbcConnection object per data source. To add these objects, drag a Data Source item to a form in a project.
- You need a top-level [RecordDefinition](#) object and a [QueryNavigator](#) 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 RecordDefinition object may be used by multiple QueryNavigator objects. The RecordDefinition object defines the structure of the a table and handles data manipulation, such as inserting rows, updating rows, and so on. The QueryNavigator objects define which rows will be pulled back from the database, how they will be inserted, and so on.

### dbANYWHERE API

- You need an additional [Session](#) component if you add another data source that is served by a different dbANYWHERE Server. Multiple servers can be accessed across multiple dbANYWHERE Servers.
- You need an additional [ConnectionInfo](#) component if you are adding another data source that is served by the same server. To add another ConnectionInfo component, open the dbNAVIGATOR window and drag the new Data Source item to your Project window.
- You need an additional [RelationViewPlus](#) component if you are adding another table from the same data source and the same server. To add another RelationViewPlus component, open the dbNAVIGATOR window and drag the new Database Table item to your Project Objects window.



## Disconnecting from a data source

```
{button Concepts,AL('dbNAVIGATOR_window',0,'')} {button See  
Also,AL('dbNAVIGATOR_pop_up_menu;Connecting_to_a_dbANYWHERE_Server;Connecting_to_a_database;Using_more_than_one_Data_Source;Disconnecting_from_a_dbANYWHERE_Server;Refreshing_dbNAVIGATOR;Creating_a_data_source',0,'',  
)}
```

You can disconnect from a data source from within the dbNAVIGATOR window: This is useful if you need to free a connection from a dbANYWHERE Server that allows a limited number of connections, for example.

1. Choose Window ► dbNAVIGATOR.
2. Expand the dbANYWHERE server item (click the +).  
The [Data Sources](#) on the server appear.
3. Right-click the Data Source item, then choose Disconnect.



## Disconnecting from a dbANYWHERE Server

```
{button Concepts,AL('dbNAVIGATOR_window',0,'')} {button See  
Also,AL('dbNAVIGATOR_pop_up_menu;Connecting_to_a_dbANYWHERE_Server;Connecting_to_a_database;Using_more_than_one_Data_Source;Disconnecting_a_database;Refreshing_dbNAVIGATOR;Creating_a_data_source',0,'')}
```

You can disconnect from a dbANYWHERE Server from within the dbNAVIGATOR window: This is useful if you need to free a connection from a dbANYWHERE Server that allows a limited number of connections, for example. Disconnecting from a dbANYWHERE server disconnects each data source that Visual Cafe is currently connected to on that server.

1. Choose Window ► dbNAVIGATOR.
2. Expand the dbANYWHERE server item (click the +).  
The [Data Sources](#) on the server appear.
3. Right-click the Data Source item, then choose Disconnect.



## Refreshing the dbNAVIGATOR window

```
{button Concepts,AL('dbNAVIGATOR_window',0,'')} {button See  
Also,AL('dbNAVIGATOR_pop_up_menu;Connecting_to_a_dbANYWHERE_Server;Connecting_to_a_database;Using_more_than_one_Data_Source;Disconnecting_a_database;Disconnecting_from_a_dbANYWHERE_Server;Creating_a_data_source',0,'')  
}
```

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

1. Choose Window ► dbNAVIGATOR.
2. Right-click a dbANYWHERE Server or [Data Source](#) item, then choose Refresh.

## Creating a data source for dbANYWHERE Server

```
{button Concepts,AL('dbNAVIGATOR_window',0,'')} {button See  
Also,AL('dbNAVIGATOR_pop_up_menu;Connecting_to_a_dbANYWHERE_Server;Connecting_to_a_database;Using_more_than_one_Data_Source;Disconnecting_a_database;Disconnecting_from_a_dbANYWHERE_Server;Refreshing_dbNAVIGATOR;Creating_a_dbAWARE_project;Wizards_using;Adding_a_table_to_your_project',0,'')}
```

If you are using dbANYWHERE Server, you may need to create a data source. There are two ways to create a data source.

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 data source. These are the kinds of small databases you might use for a dbANYWHERE server running on your local machine.

If the database type uses any of the native dbANYWHERE drivers, you must use the dbANYWHERE DataSource tool to create the data source. The native drivers for dbANYWHERE include: SQL Server, Sybase, Informix, and Oracle. These databases are more appropriate for a large system which is accessed across a network.

**Note** You must have created a database before you can create a data source for it.

## Wizards



## Using the database wizards

```
{button How  
To,AL(' Creating_a_dbAWARE_project;Creating_a_data_source_table_from_a_template;Adding_a_table_to_your_project',0,'')}  
{button See Also,AL('Projects_overview;Form_Designer_F1;Editing_a_form_howto',0,'')}
```

Visual Cafe Database Development Edition provides database wizards that help you:

- Create a new project.
- Create a new project and a data source table.
- Create a table and add it to your project.

The wizard coaches you through choosing database tables, variables in the tables, and the [database actions](#) to be performed on those tables. The applet or application is then constructed to your specifications and displays in the Form Designer. You can optionally create a table from a predefined template at the same time you create a new project.



## Creating a dbAWARE project

```
{button Concepts,AL('Wizards_using',0,'','')} {button See  
Also,AL('Creating_a_data_source_table_from_a_template;Adding_a_table_to_your_project;Projects_overview',0,'','')}
```

The Project wizard helps you create a project for an applet or application.

To start the Project wizard:

1. Choose File ► New Project.
2. Select the dbAWARE Project Wizard, and click OK.
3. Complete the wizard.

Press F1 in any wizard page to receive help on that page.





## Creating a new dbAWARE project and a data source table

```
{button Concepts,AL('Wizards_using',0,'','')} {button See  
Also,AL('Creating_a_dbAWARE_project;Adding_a_table_to_your_project;Projects_overview',0,'','')}
```

The dbAWARE Template wizard creates a new project that includes a new table in the database that is based on a Visual Cafe Database Development Edition template. After the table is created by the wizard, you can see it in dbNAVIGATOR.

1. Choose File ► New Project.
2. Choose the dbAWARE Template Wizard.

Press F1 in any wizard page to receive help on that page.



## Adding a table to your project

```
{button Concepts,AL('Wizards_using',0,'','')} {button See  
Also,AL('Creating_a_dbAWARE_project;Creating_a_data_source_table_from_a_template;Projects_overview',0,'','')}
```

The Add Table wizard helps you add a table to your project. If your project already has a [QueryNavigator](#) or [RelationViewPlus](#), you can create a [master/detail relationship](#). If you are using the [JDBC API](#), you can create [joins](#) across [forms](#).

1. In the Project window, select the [container](#) that you want to add the table to, then choose Insert ► Add Table Wizard.
2. Complete the wizard.

Press F1 in any wizard page to receive help on that page.

## Manipulating views and relationships

## Creating a view and a master/detail relationship

{button Concepts,AL('Database Development Edition\_Extension\_Overview;Wizards\_using;Master\_detail\_JDBC\_overview;QueryNavigator\_close\_overview',0,',')} {button See Also,AL('wizards\_using;Changing\_a\_view;Deleting\_a\_view',0,',')}

For the [JDBC API](#), the [QueryNavigator](#) and [RecordDefinition](#) components together represent a [view](#). For the [dbANYWHERE API](#), a [RelationViewPlus](#) component represents a view. You can also create a [master/detail relationship](#); for the JDBC API, the [join](#) can be across [forms](#).

For more information on specifying SQL statements in Visual Cafe Database Development Edition, see [Specifying SQL SELECT statements](#).

### Note

- 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 = or !=. It is not removed for all other operators, such as <. See [Managing QueryNavigator components in a Java program](#) for more information.
- If you set the Auto Start property of a detail QueryNavigator to false, you need to start the QueryNavigator programmatically before it will perform any operations. See [Managing QueryNavigator components in a Java program](#) for more information.

### To create a view by using the Project wizard

Use the Project wizard to create a new project.

#### ► [Details on this step](#)

### To create a view or detail view by using the Add Table wizard

Use the Add Table wizard to add a view to your project. The new view represents the table you specified in the wizard.


#### ► [Details on this step](#)

### To create a detail view by changing the Join properties

A [master/detail relationship](#) is the relationship between a detail [view](#) and its master view. A [QueryNavigator's](#) or [RelationViewPlus' join](#) (Join properties) defines the relationship.

1. In the Project window, select the QueryNavigator or RelationViewPlus component that will represent the detail view.
2. In the Property List, examine the Join properties.
3. Specify the master QueryNavigator in the Master Alias Name field or the master RelationView in the Parent RelationView 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 QueryNavigator 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  button to open the Join Definition dialog box.
6. Use the dialog box to specify the join definition.

#### ► [Details on this step](#)

### To create a MultiView

The [RelationViewPlus](#) class can encapsulate a MultiView object. For example, `RelationViewPlus.saveMultiView()` translates to `MultiView.save()` for the RelationViewPlus' parent MultiView. MultiView is a container class for the dbANYWHERE database transaction object. A MultiView contains a transaction's RelationViewPlus components.

To interact with a dbANYWHERE Server, you do not have to interface directly with a MultiView object. However, a MultiView object can be useful to global transactions that have a handle to a MultiView object.

## Changing a view

```
{button Concepts,AL('Database Development  
Edition_Extension_Overview;Wizards_using;Master_detail_JDBC_overview;QueryNavigator_close_overview',0,','')} {button  
See  
Also,AL('Creating_a_view;Changing_a_master/detail_relationship;Component_Properties_overview;Changing_Component_Pro  
perties_howto;Property_Inspector_using_F1;Deleting_a_view',0,','')}
```

For the [JDBC API](#), the [QueryNavigator](#) and [RecordDefinition](#) components together represent a [view](#). For the [dbANYWHERE API](#), a [RelationViewPlus](#) component represents a view.

For more information on specifying SQL statements in Visual Cafe Database Development Edition, see [Specifying SQL SELECT statements](#).

### Note

- 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 = or !=. It is not removed for all other operators, such as <. See [Managing QueryNavigator components in a Java program](#) for more information.
- If you set the Auto Start property of a detail QueryNavigator to false, you need to start the QueryNavigator programmatically before it will perform any operations. See [Managing QueryNavigator components in a Java program](#) for more information.

### To change the data model for a view

- For JDBC, in the RecordDefinition component, change the Table property.
- For the dbANYWHERE API, in the RelationViewPlus component, change the SQL property.

### To change the relationship between records in a view


- For the JDBC API, in the QueryNavigator component, change the Filter property.
- For the dbANYWHERE API, in the RelationViewPlus component, change the SQL property.

### To change a detail view by changing the Join properties

A [master/detail relationship](#) is the relationship between a detail [view](#) and its master view. You can change the master/detail relationship between records in master/detail views. A [QueryNavigator's](#) or [RelationViewPlus'](#) [join](#) (Join properties) defines the relationship.

1. In the Project window, select the QueryNavigator or RelationViewPlus component that represents the detail view.
2. In the Property List, examine the Join properties.
3. If needed, specify the master QueryNavigator in the Master Alias Name field or the master RelationView in the Parent RelationView 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 QueryNavigator 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  button to open the Join Definition dialog box.
  6. Use the dialog box to change the join definition.
- [Details on this step](#)



## Deleting a view

```
{button Concepts,AL('Database Development  
Edition_Extension_Overview;Wizards_using;Master_detail_JDBC_overview;QueryNavigator_close_overview',0,'')} {button  
See Also,AL('RelationView_component;Creating_a_view;Changing_a_view',0,'')}
```

To delete a view, simply delete the [QueryNavigator](#) or [RelationViewPlus](#) component. For the [JDBC API](#), you can delete the [RecordDefinition](#) component as well, if you are not using it with another QueryNavigator component.

## Understanding JDBC master/detail capabilities

```
{button How  
To,AL(' Creating_a_view;Changing_a_master/detail_relationship;Component_Properties_overview;Changing_Component_Prop  
erties_howto;Property_Inspector_using_F1;Changing_a_view;Deleting_a_view',0,',')} {button See Also,AL('Database  
Development Edition_Extension_Overview;Wizards_using;QueryNavigator_close_overview',0,',')}
```

[Master/detail](#) support, in its simplest form, is the ability to display and manipulate database records that are related to each other (typically through foreign keys). Associated with a given master record, there is a limited set of related detail records. When the user scrolls a master record, the application must display a set of detail records based on the new master record's values.

Master/detail provides the ability to handle these data restrictions and data filters automatically.

When you need to have a third level to the hierarchy, you can create a grand-detail record that is related to a detail record.

You can write code by hand to use the [JDBC API](#) for master/detail, but the Add Table Wizard enables you to erect a framework for master/detail that you can then customize.

In the JDBC API, master/detail [joins](#) are defined at the detail record, and use the same properties and join dialogs used with the [dbANYWHERE API](#). The basic property set necessary to establish a master/detail relationship is:

- Master navigator identifier
- Master column identifier
- Detail column identifier
- Binary operator (optional)

### Master-only data buffering

An application retrieves a set of master records by making a query to the database through the API. The set of master records returned is called the master result set. The master result set is buffered to allow forward and backward scrolling of data. To reduce memory impact on the client, and to ensure fresh data, the detail result sets are not buffered. They are resolved (discarded or saved), and a new detail result set is queried corresponding to the new master record.

### Choice of save methods

The JDBC API offers three possible save methods for you to use:

Save( )	affects the current row and all its detail rows
SaveAll( )	affects all the rows in the current view and all related detail rows
SaveAllLevels( )	begins at the topmost level and affects all rows of all related views

The following table shows the extent of a save operation in various situations:

	Save( )	SaveAll( )	SaveAllLevels( )
Parent rows saved?	None	None	All
Parent refreshed?	No	No	Yes
Current view rows saved?	Current row	All	All
Current view refreshed?	No	Yes	Yes
Detail rows saved?	All	All	All
Detail refreshed?	Yes	Yes	Yes

### Handling changed detail records

If the user makes changes in the detail result set, and then scrolls the master record, the system displays a dialog box asking the user whether to save the changes, discard the changes, or cancel the scroll operation. If the user chooses 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 SaveAllLevels method. This is to ensure data integrity.

### **Data manipulation**

When the user performs a save operation, all Data Manipulation Language ([DML](#)) operations are performed in a single transaction if possible. The DML operations are delete, insert, and update.

*Saving changes.* The system performs data changes by traversing a tree of related result sets. In order to save any changes made during a master/detail operation, the system does the deletes in the detail records, then in the related master record. The order of traversal for inserts and updates is the from the top down; for each master record, the inserts or deletes are performed first, then those of the related detail records.

*Inserting new records in the master.* When the user positions the master view on a new, unsaved row, the related detail views will not accept data. The user must first save new master row before the details will accept data. The details will not allow the user to insert or scroll records; attempting to do any of these actions will result in an exception: "Parent record has not been committed. Data changes are not permitted at this time."

### **Transactions**

Each time the user causes a DML operation to be issued to a database, the system first sends a begin transaction request to the database. This is to ensure that all calls to that particular database for the save operation occur as a unit. Once all the DML operations have been successfully issued to the particular databases involved, then all the pending transactions are committed. If any of the operations failed, then all transactions are rolled back.

All DML operations issued to the same database are performed within a single transaction if possible.

When a master/detail relationship spans more than one database, there is a scenario in which DML operations could be committed on one database and not the other(s). This situation can occur because one database supports transactions, but another does not. It can also occur if one database fails after the DML operations have occurred, but before the commit can be performed.



## Customizing Code

## Customizing Database Development Edition code

{button How  
To,AL('Adding\_a\_dbAWARE\_field\_to\_an\_existing\_form;Creating\_a\_dbAWARE\_project;Connecting\_to\_a\_dbANYWHERE\_Server;Adding\_a\_table\_to\_your\_project;Creating\_a\_data\_source\_table\_from\_a\_template',0,'')} {button See  
Also,AL('dbNAVIGATOR\_window;Visual\_Cafe\_components\_and\_containers;Wizards\_using;Mediator\_overview;Database  
Development\_Edition\_Extension\_Overview;Exception\_handling\_db\_overview;QueryNavigator\_close\_overview',0,'')}

To extend the capabilities and behavior of your Visual Cafe project, you may want to customize the code generated by Visual Cafe Database Development Edition.

For important guidelines, see [Adding code to a Java source file](#).

**Note** Database connections are closed once a JDBC object goes out of scope.

Here are some code customization tasks:

[Managing QueryNavigator components in a Java program](#)

[Closing a form when its QueryNavigator closes](#)

[Creating custom database logons](#)

[Creating a database-aware component with a mediator](#)

[Overriding DML with stored procedures](#)

[Using the JDBC transaction support features](#)

[Changing where your Java program displays errors](#)

[Handling last row, first row, and no rows exceptions](#)

[Changing Grid Cell attributes](#)

[Changing Grid column attributes](#)

[Defining automatic Grid redraw](#)

[Defining automatic Grid row numbering](#)

[Modifying the Grid toolbar](#)

[Protecting a Grid column](#)

## Managing QueryNavigator components in a Java program

```
{button How To,AL('Adding_a_dbAWARE_field_to_an_existing_form',0,'')} {button See  
Also,AL('QueryNavigator_component;Customizing_Database_Development_Edition_code;Database_Development  
Edition_Extension_Overview',0,'')}
```

The [QueryNavigator component](#) is contained by a [form](#) and provides [data binding](#) to the form. The QueryNavigator obtains and maintains data. Here are some issues you should be aware of when using this component in your Java programs.

### Adding code to manage closing the QueryNavigator and the form that contains it

When using QueryNavigator components, you need to add code to:

- Close the QueryNavigator when the form that contains it closes.  
A QueryNavigator 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 QueryNavigator when its context is destroyed; this will clean up the QueryNavigator and let garbage collection occur.
- Close the form containing a QueryNavigator when the QueryNavigator closes.  
See [Closing a form when its QueryNavigator closes](#) for more information.

### Using unique Alias Name properties in a Java program

You can use a QueryNavigator across multiple forms, for example, in a cross-form [join](#). This is possible because each QueryNavigator provides a unique name by which it can be located: the Alias Name property.

Two QueryNavigators cannot use the same Alias Name. If this is attempted, the newly added QueryNavigator takes the place of the first QueryNavigator, and garbage collection is performed. This means that if two copies of a form are created without offering a unique "per instance" Alias Name for each form's QueryNavigator, then the original form will stop functioning.

### Closing a detail form when the master form closes

A QueryNavigator might depend on a parent QueryNavigator, for example, when the QueryNavigator 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 QueryNavigator's form when its parent (master) QueryNavigator has been closed.

For example, if there is a Master form and a Detail form, and you coded the Master form to call close() on the Master QueryNavigator when the form is closed, then the following actions occur when the form is closed:

1. The Master form is closed.
2. The Master form closes the Master QueryNavigator.
3. The Master QueryNavigator notifies the Detail QueryNavigator.
4. The Detail QueryNavigator closes itself, because it is no longer valid without its parent.
5. After being notified that its QueryNavigator closed, the Detail form closes itself.

In this scenario, you must program steps 2 and 5 into your code.

### Updating a cached detail set

In a master/detail relationship, detail records are only expected to be displayed 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 = or !=. It is not removed for all other operators, such as <.

To ensure that the cache is always accurate, issue a QueryNavigator restart() call after a save operation to reissue a query to the database. Or, use the QueryNavigator 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 Auto Start property to true causes a QueryNavigator to issue its query as soon as it has been instantiated. You can turn off this behavior by setting the Auto Start property to false, which can be useful in certain cases, as described in these examples:

- You are using the QueryNavigator to insert records into a database table, but a query is never required.
- You use the QueryNavigator 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 `QueryNavigator`, at which point the detail result set will begin issuing queries automatically. The detail `QueryNavigator` does not react to scrolling of the master `QueryNavigator` until this "activation" code line is issued.

## Closing a form when its QueryNavigator closes

```
{button Concept,AL('Customizing_Database_Development_Edition_code;Database Development  
Edition_Extension_Overview;Understanding_Component_Connections_overview',0,'')} {button See  
Also,AL('QueryNavigator_component;Adding_a_dbAWARE_field_to_an_existing_form',0,'')}
```

The [QueryNavigator component](#) is contained by a [form](#) and provides [data binding](#) to the form. The QueryNavigator obtains and maintains data.

You can add code that closes a form when a QueryNavigator contained by it closes. Here is a quick way to do this by using the Interaction Wizard:

1. Select the QueryNavigator component in the Objects view of the Project window.
2. Choose Object ► Add Interaction.
3. In the Interaction Wizard, verify that the trigger QueryNavigator component appears in the Start an interaction field title; if it is wrong, 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 QueryNavigator.
6. In the Choose what you want to happen field, select an action (such as `Disable`). Choose an action that takes no parameters, so the Next button 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 five edits:

- First, Visual Cafe generates an adapter or listener implementation for the event. If one was already generated, it is used with the new interaction.
  - Second, 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](#).
  - Third, 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.
  - Fourth, an event handler is generated. If the event handler already exists, it is used.
  - Fifth, the interaction specified in the wizard is generated in the event handler.
8. Overwrite the event handler code with this one line:

```
dispose();
```

For example:

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

## Creating custom database logons

```
{button Concepts,AL('Database Development  
Edition_Extension_Overview;Customizing_Database_Development_Edition_code',0,'')} {button See  
Also,AL('PasswordDialog_component;ConnectionInfo_component',0,'')}
```

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

### JDBC API

The [JdbcConnection](#) 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 [JdbcConnection](#) 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 [Using the Class Wizard](#) 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 [JdbcConnection](#) component's Connect Failed Listener 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:

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

### dbANYWHERE API

You may want to provide a database logon screen that allows users to enter their own username and password. Using a logon dialog allows you to control and/or limit user access. Overview information is available in [How the logon process works](#) later in this topic.

Visual Cafe Database Development Edition provides a basic logon dialog, but you may wish to create your own custom logon. A custom logon can provide different components and give you more control over the details of the logon process such as which database to connect with. It also gives you more control over the look-and-feel of the logon process.

**Tip** Another way to implement your own logon dialog is to use the [PasswordDialog](#) component, which is a simple frame that asks for a username and password. Use the [PasswordDialog](#) to obtain the username and password, and pass those values to the [ConnectionInfo](#) component by using the [setUser\(\)](#) and [setPassword\(\)](#) methods shown on lines 12 and 13 of the "how the logon process works" code segment below. Creating a custom logon can get complicated when you want to handle invalid user input. In that case, you can create a sophisticated logon dialog by using the [Logon](#) interface as described below.

A custom logon class must implement the [Logon](#) interface. The [Logon](#) interface is defined in [Logon.java](#) located in the [symantec.itools.db.pro](#) package.

To understand how to create a custom logon, review the default [LogonFrame](#) source code. To review the source code, follow these steps:

1. Open or create a database project.

#### ► [Details on this step](#)

2. Open the Class Browser window (Window ► Class Browser or CTRL+SHIFT+C)
3. Open the [symantec.itools.db.awt](#) package (at the bottom of the list).
4. Right-click on the [LogonFrame](#) and select Go to Source.

The fastest way to create your own custom logon is to copy the displayed source code into a new file and make your changes from there.

5. Select File ► New File and enter the following code:

```
1: public class MyLogonFrame extends symantec.itools.db.awt.LogonFrame
2: {
3:     public MyLogonFrame() {
4:         setBackground(java.awt.Color.lightGray);
5:     }
6: }
```

This code produces the same logon box as the PasswordDialog component, but has a light gray background color instead of the default white.

6. The Session object will call the installed Logon object's logonFailed method whenever dbANYWHERE fails to connect to a database.

Logon.logonFailed returns a boolean indicating whether dbANYWHERE should continue to try to connect to the database.

The default behavior for symantec.itools.db.awt.LogonFrame.logonFailed method is to display a Frame that prompts the user for a new name and password. If the user clicks the OK button, LogonFrame will set the ConnectionInfo's username and password to the new values and return the boolean true so dbANYWHERE will attempt to connect again. If the user clicks Cancel, a boolean false is returned. dbANYWHERE will, in this case, throw a SQLException back to the method that initiated the database connection. LogonFrame ignores the retries parameter.

You may want to modify LogonFrame's logonFailed implementation. For instance:

- You may want to limit the number of attempts that the user has to connect to a database.
- You may want to validate the username and password that was typed in.

You will have to implement the logonFailed method in your MyLogonFrame class:

```
public boolean logonFailed(ConnectionInfo conn, int retries)
{
    //Examine the retries parameter and return false if it is
    //greater than some predefine value
    if (retries > 5)
        return false;
    //Call LogonFrame's implementation
    boolean continue = super.logonFailed(conn,retries);

    //If user wants to continue, you may want to validate the new
    //username and password
    if (continue)
    {
        String username = conn.getUser();
        String password = conn.getPassword();
        //Validate username and password
        //return true if validation passed or false if it failed
    }
    return continue ;
}
```

7. Save the file as MyLogonFrame.java. Be sure to check the Add to Project checkbox in the lower left corner of the Save As dialog box.
8. In your main applet, the project needs to use your new logon frame. Go to the line that instantiates the logon frame (line 4 in the previous code example) and change the reference of symantec.itools.db.awt.LogonFrame to MyLogonFrame.
9. In order for the logon frame to appear at all, you must comment out and remove the user name and password. Lines 12 and 13 of the code snippet below show where those items are. Remove them from the property lists as well (they are properties of the ConnectionInfo component). You must remove both sources independently, since Visual Cafe does not currently support two-way development on the ConnectionInfo component.
10. Execute your project.

### How the logon process works

The following code segment illustrates the relatively simple steps involved in the logon process. The dialog for this logon follows the code. Code notes follow the illustration.

```
1 try {
2     symantec = new symantec.itools.db.pro.Session("dbaw://localhost:8889",
3     java.beans.Beans.isDesignTime());
4     symantecLogonFrame = new symantec.itools.db.awt.LogonFrame();
```

```

5      symantec.setLogonObject(symantecLogonFrame);
6  } catch (symjava.sql.SQLException e) {
7      System.out.println(e.getMessage());
8      return;
9  }
10 SQL_Anywhere_5_0_Sample = new
11     symantec.itools.db.pro.ConnectionInfo("SQL Anywhere 5.0 Sample");
12 SQL_Anywhere_5_0_Sample.setUser("dba");
13 SQL_Anywhere_5_0_Sample.setPassword("sql");
14 SQL_Anywhere_5_0_Sample.setAutoDisconnect(false);
15 try {
16     symantec.itools.db.pro.Request DBA_packagesRequest = new
17         symantec.itools.db.pro.Request(symantec,SQL_Anywhere_5_0_Sample);
18     . . .
19 } catch (java.sql.SQLException e) {
20     System.out.println(e.getMessage());
21     return;
22 }

```



#### Code Notes

On line 4, a LogonFrame object is instantiated. The LogonFrame displays whenever dbANYWHERE fails to connect to a database. This may occur because

- the ConnectionInfo object has an empty or invalid username or password.
- the username or password typed in the User Authentication Frame previously by the enduser was invalid.

On line 5, the LogonFrame is associated with the Session component, which means that only one type of logon object is associated with a given dbANYWHERE Session.

Lines 10 through 14 instantiate and populate the ConnectionInfo component. In particular, note lines 12 and 13 that set the username and password for the ConnectionInfo component.

Lines 16 and 17 (all one line, broken into two) is where the first access to the database occurs. The parameters for the Request object are the Session and ConnectionInfo components.

If lines 12 and 13 were commented out, the LogonFrame object would automatically appear to ask for a username and password. These two fields are then be passed back to the logon process of the Request object shown in step 4.





## Overriding DML with stored procedures

```
{button Concepts,AL('Database Development  
Edition_Extension_Overview;Customizing_Database_Development_Edition_code',0,'','')} {button See Also,AL('Creating a  
dbAWARE project;Creating a database table;Adding a table to your  
project;Projects_overview;Form_Designer_F1;Editing_a_form_howto',0,'','')}
```

For the [JDBC API](#), you can modify code in the [RecordDefinition](#) component of a project to override [DML](#) with [stored procedures](#).

1. Make sure a RecordDefinition component is in your project. If not, add it.
2. Open the Java source file for this RecordDefinition component.
3. Override one or more of the following methods:
  - `getStatement`
  - `getInsertStatement`
  - `getUpdate`
  - `getDelete`
  - `getSave`

These methods return a statement object (`java.sql.statement`) used to perform various operations, such as insert, update, save, and so on.



## Using the JDBC transaction support features

```
{button Concepts,AL('Database Development  
Edition_Extension_Overview;Customizing_Database_Development_Edition_code',0,'','')} {button See  
Also,AL('PasswordDialog_component;ConnectionInfo_component',0,'','')}
```

You can override transactions. You can also set the transaction isolation level by modifying code for the [ConnectionManager](#) component:

```
jdbcConnection.setTransactionIsolation(Value)
```

The *isolation level* defines degree to which the operations in one transaction are visible to the operations in a concurrent transaction. Isolation levels prevent some or all inconsistent behavior and can be different for each connection. All isolation levels guarantee that each transaction will execute completely or not at all, and that no updates will be lost. This ensures recoverability at all times, regardless of the isolation level.

There are three types of inconsistency that can occur during the execution of concurrent transactions:

**Dirty read** — Occurs when one transaction modifies a row then another transaction reads that row before the first transaction performs a COMMIT. If the first transaction performs a ROLLBACK, the second transaction has read a row that was never committed.

**Non-repeatable read** — Occurs when one transaction reads a row then another transaction modifies or deletes the row and performs a COMMIT. If the first transaction attempts to read the same row again, the row has been changed or deleted.

**Phantom row** — Occurs when one transaction reads a set of rows that satisfy some condition, then another transaction executes an INSERT, or an UPDATE (that generates one or more rows that satisfy the condition used by the first transaction) and then performs a COMMIT. The first transaction then repeats the initial read and obtains a different set of rows.

Here are the types of transaction isolation levels:

TRANSACTION\_NONE — Transactions are not supported.

TRANSACTION\_READ\_UNCOMMITTED — Lets dirty reads, non-repeatable reads, and phantom reads occur.

TRANSACTION\_READ\_COMMITTED — Prevents dirty reads, but not non-repeatable reads and phantom reads.

TRANSACTION\_REPEATABLE\_READ — Prevents dirty reads and non-repeatable reads, but not phantom reads.

TRANSACTION\_SERIALIZABLE — Prevents dirty reads, non-repeatable reads, and phantom reads.

## Understanding exception handling in Database Development Edition code

```
{button How
To,AL('Creating_a_dbAWARE_project;Customizing_Database_Development_Edition_code;Changing_where_program_errors_di
splay_howto;Handling_row_exceptions_howto',0,'')} {button See
Also,AL('dbNAVIGATOR_window;Visual_Cafe_components_and_containers;Wizards_using;Mediator_overview;Creating_a_Mul
tiView;Database_Development
Edition_Extension_Overview;Connecting_to_a_dbANYWHERE_Server;Adding_a_table_to_your_project;Creating_a_data_sourc
e_table_from_a_template',0,'')}
```

For the [JDBC API](#), the [QueryNavigator component](#) registers itself as a [listener](#) for all kind of [exceptions](#) encountered within all classes handled by QueryNavigator. This code is in the constructor of QueryNavigator:

```
public QueryNavigator(){

    ...

    addExceptionEventListener(this);
}
```

QueryNavigator handles the encountered exceptions by implementing:

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

So 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 unexpected 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

```
{button Concepts,AL('Visual_Cafe_components_and_containers;Database Development  
Edition_Extension_Overview;Exception_handling_db_overview',0,'')} {button See  
Also,AL('Creating_a_dbAWARE_project;Connecting_to_a_dbANYWHERE_Server;Adding_a_table_to_your_project;Creating_a  
_data_source_table_from_a_template',0,'')}
```

For the [JDBC API](#), to change where errors are displayed, you need to register a [listener](#) for [QueryNavigator](#) in the Java program code. For example, if you want to show errors in a pop-up window, add the following code to the Java program:

```
public void init() {  
  
    ...  
  
    queryNavigator1.addExceptionEventListener(window);  
}
```

**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 you can tune your [event handler](#) according to the type of the [exception](#).

## Handling last row, first row, and no rows exceptions

```
{button Concepts,AL('Visual_Cafe_components_and_containers;Database Development  
Edition_Extension_Overview;Exception_handling_db_overview',0,'')} {button See  
Also,AL('Creating_a_dbAWARE_project;Connecting_to_a_dbANYWHERE_Server;Adding_a_table_to_your_project;Creating_a  
_data_source_table_from_a_template',0,'')}
```

Here are some special cases:

- The end of the RecordSet was reached and Next button is clicked.  
The [QueryNavigator](#) fires an ExceptionEvent of the EXCEPTION\_LASTROW type.
- The beginning of the RecordSet was reached and Previous button is clicked.  
The QueryNavigator fires an ExceptionEvent of the EXCEPTION\_FIRSTROW type.
- The RecordSet has no rows.  
The QueryNavigator fires an ExceptionEvent of the EXCEPTION\_NOROWS type.

For the [JDBC API](#), by default the QueryNavigator 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 QueryNavigator.

Here is sample Java program code:

```
public int init() {  
  
    ...  
  
    queryNavigator1.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  
        queryNavigator1.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);  
    }  
}
```

## Mediators



## Using mediators

```
{button How To,AL(' Mediator_with_non_database_component;Mediator_making_component_database_aware',0,'','')} {button  
See Also,AL(' Creating a dbAWARE project;Projects_overview;Form_Designer_F1;Editing_a_form_howto',0,'','')}
```

A [mediator](#) lets you make a [component](#) database-aware. It is like a bridge between the component and the [QueryNavigator](#) or [RelationViewPlus](#) components.

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 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 Environment Options, Database view.

Here is more information on tasks you can perform with a mediator:

[Using a non-database-aware component with a mediator](#)

[Creating a database-aware component with a mediator](#)



## Using a non-database-aware component with a mediator

```
{button Concepts,AL('Mediator_overview',0,'')} {button See Also,AL('Creating a dbAWARE  
project;Projects_overview;Form_Designer_F1;Editing_a_form_howto',0,'')}
```

At design time, you might want to [bind](#) a non-database-aware [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. See [Creating a database-aware component with a mediator](#) for more information.

1. Add a non-database-aware component to a form.

### ► [Details on this step](#)

2. Add a Mediator component to a form.

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

3. Set the properties of the Mediator component.

Property	Description
Output	Name of the non-database-aware component.
SetMethods	String array property made of the methods the mediator should use to provide the non-database-aware component with data (this depends on the methods used for the component); each method should be of this format: <i>method-name(value [, row] [, col] )</i> , for example { "setLabel (Value) " }. The method or methods are enclosed by { " and " }, and delimited with a comma.
GetMethods	String array property for obtaining the data from the component (this depends on the methods used for the component); each method should be of this format: <i>method-name( [row] [, col] )</i> , for example { "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 <a href="#">QueryNavigator</a> or the object name for the <a href="#">RelationViewPlus</a> , followed by the column name, number, or both (ALL is a valid value). For example, queryNavigator1@customer@last_name %5, where 5 is the number of rows to display.

## Creating a database-aware component with a mediator

{button Concepts,AL('Mediator\_overview',0,'')} {button See Also,AL('Creating a dbAWARE project;Creating a database table;Adding a table to your project;Projects\_overview;Form\_Designer\_F1;Editing\_a\_form\_howto',0,'')}

To create a database-aware [component](#), the [mediator](#) is defined within the component. Alternatively, at design time you can [bind](#) a non-database-aware component. See [Using a non-database-aware component with a mediator](#) for more information.

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 setDataBinding(String binding) {
        m_Mediator.setDataBinding(binding);
    }

    public String getDataBinding() {
        return m_Mediator.getDataBinding();
    }
}
```

Follow these steps to add a mediator to a non-database-aware component. For more information on creating a [JavaBeans component](#), see [Creating JavaBeans components in Visual Cafe](#).

1. Start with a project that contains the Java file for the JavaBeans component.
2. Add a mediator to the component by adding this line of code:  
`private Mediator m_Mediator = new Mediator();`
3. 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);
```

Property	Description
Output	Name of the non-database-aware component.
SetMethods	String array property made of the methods the mediator should use to provide the non-database-aware component with data (this depends on the methods used for the component); each method should be of this format: <i>method-name</i> ( <b>value</b> [, <b>row</b> ] [, <b>col</b> ] ), for example { "setLabel(Value)" }. The method or methods are enclosed by { " and " }, and delimited with a comma.
GetMethods	String array property for obtaining the data from the component (this depends on the methods used for the component); each method should be of this format: <i>method-name</i> ( [ <b>row</b> ] [, <b>col</b> ] ), for example

`{ "getLabel() " }`. The method or methods are enclosed by { " and " }, and delimited with a comma.

4. Add the methods for setting the data binding property:

```
public void setDataBinding(String binding) {  
    m_Mediator.setDataBinding(binding);  
}  
  
public String getDataBinding() {  
    return m_Mediator.getDataBinding();  
}
```

These methods should follow the standard coding guidelines for properties. You should expose the data binding property in the BeanInfo so your users can set it.

## Deployment



## Deploying database-aware Java programs

{button How

To,AL('Creating\_a\_dbAWARE\_project;Connecting\_to\_a\_dbANYWHERE\_Server;Adding\_a\_table\_to\_your\_project;Creating\_a\_data\_source\_table\_from\_a\_template',0,'')} {button See

Also,AL('dbNAVIGATOR\_window;Visual\_Cafe\_components\_and\_containers;Wizards\_using;Mediator\_overview;Creating\_a\_MultiView;Database\_Development\_Edition\_Extension\_Overview;Customizing\_Database\_Development\_Edition\_code',0,''))}

Deploying applets and applications developed with Visual Cafe Database Development Edition is the same as deploying Visual Cafe applets and applications. See [Deploying applets](#) and [Deploying applications](#) for more information.

Note that there are additional classes for the Database Development Edition. The directory bin\components contains databind.jar, dbaw\_awt.jar, and symbeans.jar. There is also this ZIP file: java\lib\dbaw.zip.

Obsolete

## Creating a master/detail relationship using JDBC

```
{button Concepts,AL('Database Development
Edition_Extension_Overview;Customizing_Database_Development_Edition_code',0,'')} {button See
Also,AL('master/detail_names_howto;Component_Properties_overview;Changing_Component_Properties_howto;Property_Insp
ector_using_F1;Adding_Code_to_a_Java_Source_File_howto',0,'')}
```

A [master/detail relationship](#) is the relationship between a detail [view](#) and its master view.

To make the master/detail relationship using JDBC, you must manually setup the relationship in your project. A sample of a project using master/detail can be found in Samples\Symantec\Applets\JDBC\MasterDetail. The following procedure walks you through the steps to build a copy of the sample and uses it to help explain the details. You can adapt this procedure to create your own applets and applications with master/detail relationships that use JDBC

### To create a master/detail relationship

1. Choose File ► New Project  
► dbAWARE Applet.
2. Follow the instructions in the wizard to link to the Sybase SQLAnywhere database through dbANYWHERE.
3. Follow the instructions in the wizard to select Department as your database table. This table will be used as the master view.
4. After the applet is generated, choose Project ► Execute to make sure the applet runs correctly.
5. Once the basic applet has been verified, quit the Applet Viewer and close the Form Designer.
6. Choose Insert ► Form  
► Dialog. This dialog box will contain the detail view. You do not have to create a separate form to contain the detail view. If you want your detail view on the same form as the master view, open the master view form before proceeding to the next step.
7. With the dialog box active, choose Insert ► Add Table Wizard.
8. Follow the instructions in the wizard to add the Employee database table as a grid. This table will be used as the detail view. Adding the table as a grid is optional.
9. Close the dialog box.
10. Save your project. Do not overwrite the project in the JDBC directory.
11. Copy Relationship.java and DetailQueryNavigator.java to your project directory from Samples\Symantec\Applets\JDBC\MasterDetail, then choose Insert ► Files to add these files to the project.
12. Edit the detail file, DBA\_employeeRecord, and copy the following code from Samples\Symantec\Applets\JDBC\MasterDetail\DBA\_employeeRecord.java into the file. Place the code in the INIT method outside the INIT\_CONTROLS section.

```
m_Join = new Vector();

// Add one Relationship object to the m_Join vector as in the example
// below for every column to column relationship in your join.

m_Join.addElement(new Relationship(
    "DBA_departmentNavigatorAlias",    // master navigator alias
    1,                                // master column index
    "dept_id",                        // master column name
    5,                                // detail column index
    "dept_id"));                      // detail column name
```
13. Modify the code to add the following join information: Master Navigator Alias, Master Column Index, Master Column Name, Detail Column Index, and Detail Column Name. Each of these items can be obtained from viewing various properties.  
To locate the join information:  
► master/detail\_names\_howto [Details on this step](#) master/detail\_names\_howto
14. Copy the following code from MasterDetail\DBA\_employeeRecord.java to the detail.java file. Paste the code outside of any sections.

```
private static Vector m_Join = null;

public Enumeration querySimilarObjects(String whereClause,
    String[] sortOrder) throws SQLException {
    StringBuffer extendedWhereClause = new StringBuffer("");
```

```

        if (whereClause != null) {
            StringTokenizer tokens = new StringTokenizer(whereClause, " ");
            if (tokens.countTokens() >= 2) {
                String token1 = tokens.nextToken();
                if (!token1.equalsIgnoreCase("where")) {
                    extendedWhereClause.append(" where ");
                }
            }
            else {
                extendedWhereClause.append(" where ");
            }
            extendedWhereClause.append(whereClause);

            for (int index = 0; index < m_Join.size(); index++) {
                extendedWhereClause.append(" and ");
                Relationship join = (Relationship)m_Join.elementAt(index);
                extendedWhereClause.append(join.m_DetailColumnName);
                extendedWhereClause.append(" = ?");
            }
        }
        else {
            extendedWhereClause.append(" where ");
            for (int index = 0; index < m_Join.size(); index++) {
                if (index != 0) {
                    extendedWhereClause.append(" and ");
                }
                extendedWhereClause.append(((Relationship)m_Join.elementAt(index)).m_DetailColumnName);
                extendedWhereClause.append(" = ?");
            }
        }

        return super.querySimilarObjects(extendedWhereClause.toString(), sortOrder);
    }

    public void setQueryParameters(PreparedStatement stmt)
    {
        DataModel model = (DataModel)getDataModel();
        for (int index = 0; index < m_Join.size(); index++) {
            // ((Relationship)m_Join.elementAt(index)).m_MasterColumnValue =
            // m_Master.getValue(0, ((Relationship)m_Join.elementAt(index)).m_MasterColumnIndex);
            try {
                setObjectInStatement(stmt,
                    ((Relationship)m_Join.elementAt(index)).m_MasterColumnValue, index, 0, 0);
            }
            catch (SQLException ex) {
                throw new RuntimeException(ex.getMessage());
            }
        }
    }
}

```

**Note** Make any changes you want to the [QueryNavigator](#) object before performing the following steps. After making changes to dialog1.java, the QueryNavigator object will no longer be displayed in the Objects Tab of the Project Window and you will no longer be able to modify properties of the QueryNavigator object through the Property List. If you need to modify any of the properties of the QueryNavigator object in the future, you will need to undo the steps below, make the changes, and then redo the following steps.

15. Replace the following line of code in dialog1.java

```
DBA_employeeNavigator = new symantec.itools.db.beans.jdbc.QueryNavigator();
```

with

```
DBA_employeeNavigator = new DetailQueryNavigator("DBA_departmentNavigatorAlias");
```

where *DBA\_departmentNavigatorAlias* is the master alias name.

16. Move the following code outside the INIT\_CONTROLS and DECLARE\_CONTROLS sections.

```

DBA_employeeNavigator = new DetailQueryNavigator("DBA_departmentNavigatorAlias");
DBA_employeeNavigator.setAutoStart(true);
DBA_employeeNavigator.setClassName("DBA_employeeRecord");
DBA_employeeNavigator.setAliasName("DBA_employeeNavigatorAlias");
//$$ DBA_employeeNavigator.move(0,0);

```



and place it below the following code:

```
public Dialog1(Frame parent, boolean modal) {  
    super(parent, modal);
```

17. Move this line outside of DECLARE\_CONTROLS

```
symantec.itools.db.beans.jdbc.QueryNavigator DBA_employeeNavigator;
```

18. Move Panel2 in the master view (the panel that contains the record number label and record state label) below all the dbAWARE components in Panel1 in the master view.
19. Open the applet form in the Form Designer.
20. Add a button to open the Detail dialog box by placing a standard button on the form.
21. Use the Interaction Wizard to create an interaction between the button and Dialog1.
22. Select Create and Show as Nonmodal from the Choose What You Want to Happen field.

You can now execute the applet and make sure the master/detail relationship is functional.

## Locating the join information needed to set up a relationship using JDBC

```
{button Concepts,AL('Database Development
Edition_Extension_Overview;Customizing_Database_Development_Edition_code'0,','')} {button See
Also,AL('JDBCmaster_detail_relationship;Component_Properties_overview;Changing_Component_Properties_Property_I
nspector_using_F1;Adding_Code_to_a_Java_Source_File_howto',0,)}
```

When creating a master/detail relationship using the [JDBC API](#), you must add code to your detail file that sets up the relationship. After adding the code you must modify it so it reflects the [join](#) information from your database tables.

```
// Add one Relationship object to the m_Join vector as in the example
// below for every column to column relationship in your join.
```

```
m_Join.addElement(new Relationship(
    "DBA_departmentNavigatorAlias",    // master navigator alias
    1,                                // master column index
    "dept_id",                        // master column name
    5,                                // detail column index
    "dept_id"));                      // detail column name
```

Note the sample included with Visual Cafe. The five items you need to get information about are:

- Master Navigator Alias
- Master Column Index
- Master Column Name
- Detail Column Index
- Detail Column Name

Information about these items can be obtained by looking at various properties in the Property List.

### To find the Master Navigator Alias

1. Choose the [QueryNavigator](#) in the master form. This file will have a name like *tablenameNavigator* where *tablename* is the name of the master view.
2. Look at the Alias Name in the Property List. This name is the Master Navigator Alias Name.

### To find the Master Column Index

1. Choose the QueryNavigator in the master form.
2. Look at the Class Name in the Property List. This is the name of the Record Definition file you need to select in the Object Tab of the Project Window.
3. Select the file by name in the Object Tab.
4. Double-click the Table Name property in the Property List.
5. The xx Table displays.
6. Look in the Chosen column. The columnindex numbers are not displayed. Each field in the Chosen column relates to a columnindex number starting from 1 at the top and incrementing by one for each additional field.

### To find the Master Column Name

1. Choose Applet1 from the Object Tab of the Project Window.
2. Look at the Class Name in the Property List. This is the name of the Record Definition file you need to select in the Object Tab of the Project Window.
3. Select the file by name in the Object Tab.
4. Double-click the Table Name property in the Property List.
5. Look in the Chosen column. Each field in the Chosen column is part of the master view. Look for the field name that contains data that corresponds to a field in the detail view.

### To find the DetailColumn Index

1. Select the detail file.
2. Double-click the Table Name property in the Property List.

3. Look in the Chosen column. The columnindex numbers are not displayed. Each field in the Chosen column relates to a columnindex number starting from 1 at the top and incrementing by one for each additional field.

**To find the DetailColumn Name**

1. Select the detail file.
2. Double-click the Table Name property in the Property List.
3. Look in the Chosen column. Each field in the Chosen column is part of the detail view. Look for the field name that contains data that corresponds to the field for the master view.



## Creating a MultiView

```
{button How To,AL(' Customizing_Database_Development_Edition_code',0,'')} {button See  
Also,AL('RelationView_component',0,'')}
```

The [RelationViewPlus](#) class can encapsulate a MultiView object. For example, `RelationViewPlus.saveMultiView( )` translates to `MultiView.save( )` for the RelationView's parent MultiView. MultiView is a container class for the dbANYWHERE database transaction object. A MultiView contains a transaction's RelationViews.


To interact with a dbANYWHERE Server, you do not have to interface directly with a MultiView object. However, a MultiView object can be useful to global transactions that have a handle to a MultiView object.



## Creating a master/detail relationship

```
{button Concepts,AL('Database Development Edition_Extension_Overview;Wizards_using',0,'','')} {button See  
Also,AL('Creating_a_view;Changing_a_view;Component_Properties_overview;Changing_Component_Properties_howto;Proper  
ty_Inspector_using_F1',0,'','')}
```

<<Need to update this for 2.5>> A [master/detail relationship](#) is the relationship between a detail [view](#) and its master view. A [QueryNavigator's](#) or [RelationView's](#) [join](#) (Join properties) defines the relationship.

1. In the Form Designer, click the QueryNavigator or RelationView component that will represent the detail view.
  2. Click in the Join Columns property field.
  3. Click the  button to open the Join Definition dialog box.
  4. Use the dialog box to change the join definition.
- [Details on this step](#)

be performed.

## What happens when a QueryNavigator closes?

```
{button How To,AL('Adding_a_dbAWARE_field_to_an_existing_form',0,'')} {button See  
Also,AL('QueryNavigator_component;Customizing_Database_Development_Edition_code;Database Development  
Edition_Extension_Overview',0,'')}
```

When a [QueryNavigator](#) closes, the following actions must occur:

- If that QueryNavigator has dirty data, that data must be resolved.
- If that QueryNavigator has details, those details must be closed as well.

It is good programming practice to close a QueryNavigator when its context is destroyed. In case a programmer does not do this, however, if a [form](#) is destroyed and recreated, Visual Cafe closes the first QueryNavigator before the second one replaces it.

This means that if you destroy a form with a detail QueryNavigator in it and you recreate the form, you end up with a functioning detail QueryNavigator; but any grand detail queries will have been closed.

Following is more detailed information on what occurs when a QueryNavigator closes:

1. The act of closing or being closed is an event, following the standard event model. Here is how the close event notification spreads through the related objects in a master detail application:
  - 1) Form 1 notifies QueryNavigator 1.
  - 2) QueryNavigator 1 notifies QueryNavigatorLink 1.
  - 3) QueryNavigatorLink 1 notifies QuerySynchronizer.
  - 4) QueryNavigatorLink 1 notifies QueryNavigatorLink 2.
  - 5) QueryNavigatorLink 2 notifies QuerySynchronizer.
  - 6) QueryNavigatorLink 2 notifies QueryNavigator 2.
  - 7) QueryNavigator 2 notifies Form 2.
2. QueryNavigatorLinks and QueryNavigators register as listeners to each other. To avoid recursion, they deregister themselves as listeners of each other before they re-announce the event.
3. QuerySynchronizers register as listeners of QueryNavigatorLinks.
4. QueryNavigators and QueryNavigatorLinks do nothing more than announce the event to their registered listeners.
5. QuerySynchronizers remove QueryNavigatorLinks from the global list, remove QueryNavigatorLinks from their parent if they have parents, and resolve dirty data.
6. Forms can and should register as listeners of queries.

Here are some other considerations when using QueryNavigator components:

- When you add a record that does not match the current detail set, the record will appear in the detail set under certain conditions. For example, if a record does not fit the current detail set and you press Save, the record is removed from the detail result set only if the join operator is = or !=. It is not removed for all other operators, such as <. You should use Save All or Reset if you want the detail set to always be correct. It impacts deleting, updating, and inserting records in a detail set, and the key of the detail record does not actually fit the master.
- Detail QueryNavigators are not expected to perform any operations until they have been started. This means that if the detail QueryNavigator has the auto-start property set to false, you must call the QN.restart operation before the QueryNavigator will react to scrolling of the master QueryNavigator. This allows you to develop detail QueryNavigators that can be hidden and not functional until this "activation" code line is issued.

## Understanding the QueryNavigator component

```
{button How To,AL('Adding_a_dbAWARE_field_to_an_existing_form',0,'')} {button See  
Also,AL('QueryNavigator_component;Customizing_Database_Development_Edition_code;Database Development  
Edition_Extension_Overview',0,'')}
```

[Data binding](#) is provided to a [form](#) through the [QueryNavigator component](#). When the form is created, QueryNavigators on the form are instantiated to obtain and maintain data. When the form is closed, the QueryNavigator should also be closed. Visual Cafe does not do this automatically, because it is possible for the QueryNavigator to be used across multiple forms. This is possible because each QueryNavigator provides a unique name by which it can be located: the Alias Name property.

Two QueryNavigators cannot use the same Alias Name. If this is attempted, Visual Cafe removes the first QueryNavigator and the newly added QueryNavigator takes its place. This means that if two copies of a form are created without offering a unique "per instance" Alias Name for that form's QueryNavigators, then the first form will stop functioning. The QueryNavigator is automatically cleaned up and garbage collection is performed.

### QueryNavigator components and master/detail relationships

A QueryNavigator might depend on a parent QueryNavigator. This is the case when the QueryNavigator is the detail in a [master/detail relationship](#). In this scenario, an event system is provided that can be used to close the QueryNavigator's form when its parent (master) QueryNavigator has been closed.

It is good programming practice to close a QueryNavigator when its context is destroyed. This will clean up the QueryNavigator and allow it to be garbage collected. As mentioned previously, if a programmer does not do this, the first QueryNavigator is automatically destroyed when the second one replaces it.

The following code shows how to close a Frame containing a QueryNavigator that has been closed. (Visual Cafe does not generate this code automatically.)

```
init {  
    //{  
    .....  
    //}  
    QueryNavigator.addPropertyChangeListener(aPropertyChange);  
}  
  
SymPropertyChangeListener aPropertyChange = new SymPropertyChangeListener();  
class SymPropertyChangeListener implements java.beans.PropertyChangeListener {  
    public void propertyChange(java.beans.PropertyChangeEvent evt) {  
        if(evt.getPropertyName().equals("close")){  
            // This method is automatically created when you add a frame to the GUI.  
            // It hides and disposes of the frame, then quits the app.  
            Frm1_WindowClosing(null);  
        }  
    }  
}
```

Following is more detailed information on what occurs when a QueryNavigator closes. In this example, there is a Master form and a Detail form. The master form is coded to call close() on the master QueryNavigator when the form is closed.

1. The Master form is closed and it closes the Master QueryNavigator.
2. The Master QueryNavigator notifies the Detail QueryNavigator.
3. The Detail QueryNavigator closes itself, since it is no longer valid without its parent.
4. The Detail form is notified that its QueryNavigator closed and it closes itself.

In a master/detail relationship, detail records are only expected to be displayed 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 = or !=. It is not removed for all other operators, such as <.

It is possible to ensure that the cache is always accurate by issuing a restart() after each save. This reissues a query to the database. To ensure that the cache is always accurate, issue a restart() after a save operation to reissue a query to the database. Or, use the saveAll() method, which saves all of the changes in the cache and restarts the result set.

### Enabling a QueryNavigator's query

Setting the Auto Start property to true causes a QueryNavigator to issue its query as soon as it has been instantiated. This common behavior can be turned off by setting the Auto Start property to false, which can be useful in certain cases, as described in these examples:

- The QueryNavigator is being used to insert records into a database table, but a query is never required.
- The QueryNavigator is defined to display a detail result set, but the detail set is disabled until you issue a restart() method. In this 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 QueryNavigator, at which point the detail result set will begin issuing queries automatically. The detail QueryNavigator will not react to scrolling of the master QueryNavigator until this "activation" code line is issued.



## Top of ctaskpro.doc

This is the top of the ctask.doc file. This file contains procedures for working with components from the Component Library. This file is new to the help project for release 1.01d

11/97: Rewrote for JDBC subprotocols ODBC and dbAW. Also corrected some display and naming problems, and added browse sequences.

**Main index entry**

Please click on a sub index entry for topic information.

**Main index entry**

Please click on a sub index entry for topic information.

## Grids

## Adding a Grid component

```
{button Concepts,AL('Database Development Edition_Extension_Overview;dbNAVIGATOR_window',0,'')} {button See  
Also,AL('Grid_component;RelationView_component;dbAWARE_Binding_property;dbAWARE_Join_property;Adding_a_Grid_co  
mponent_to_a_form_as_a_detail_view;Changing_Grid_Cell_Attributes;Changing_Grid_column_attributes;Defining_automatic_G  
rid_row_numbering;Defining_automatic_Grid_redraw;Modifying_the_Grid_toolbar')}
```

To add a Grid component, follow these steps:

1. Drag the table icon, from the dbNAVIGATOR window, onto the form.  
Visual Cafe then automatically creates all other components that are needed to define the database connection; such as the [ConnectionManager](#) and [JdbcConnection](#) components or the [Session](#) and [ConnectionInfo](#) components.  
The columns display in the Grid, from left to right, in the same order that they appear in dbNAVIGATOR .
2. Drag the Grid component from the Component Library or Palette to add it to your form.  
The Grid is empty. You do not see the actual data until run time.
3. Size the Grid as appropriate.  
If query columns do not fit within the Grid component, a scroll slider is provided.
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** Column heading text can be customized in code only.
5. Set the Grid component's Data Binding property, which includes the [QueryNavigator](#) alias name or [RelationViewPlus](#) object name of the component that was created in step 1. You can leave the Field Name blank to get all fields, or specify one or more fields delimited with a comma (.).  
**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 QueryNavigator you enter does not have to be contained by the same form as the component.
6. To run the form, choose Project ► Execute.  
The Grid component uses the database column names as the Grid column headings.  
You can narrow the query by adjusting the SELECT statement. See [Specifying SQL SELECT statements](#) for more information.

## Changing Grid cell attributes

```
{button Concepts,AL('Database Development  
Edition_Extension_Overview;Customizing_Database_Development_Edition_code',0,'')} {button See  
Also,AL('Grid_component;Adding_a_Grid_component_to_a_form_as_a_detail_view;Changing_Grid_column_attributes;Changin  
g_Grid_column_attributes;Defining_automatic_Grid_row_numbering;Defining_automatic_Grid_redraw;Modifying_the_Grid_toolb  
ar;Adding_Code_to_a_Java_Source_File_howto',0,'')}
```

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 component's source code. Place any custom source code after the INIT\_CONTROLS code block.

### 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 code.

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

**Note** Cell row and cell column numbers are one-based, not zero-based.

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

You can also change the color 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);
```

### Fonts

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

```
grid1.setCellFont(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

```
{button Concepts,AL('Database Development  
Edition_Extension_Overview;Customizing_Database_Development_Edition_code',0,'')} {button See  
Also,AL('Grid_component;Adding_a_Grid_component_to_a_form_as_a_detail_view;Changing_Grid_Cell_Attributes;Protecting_a  
_Grid_column;Defining_automatic_Grid_row_numbering;Defining_automatic_Grid_redraw;Modifying_the_Grid_toolbar;Adding_  
Code_to_a_Java_Source_File_howto',0,)}
```

You can change attributes of Grid columns, such as heading names, column alignments, heading colors, and heading fonts, by enhancing the Grid component's Java code. Place any custom source code after the INIT\_CONTROLS code block.

### Changing column headings

You can change 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:

```
grid1.setHeadingColors (1, Color.black, Color.white);  
grid1.setHeadingFont (new Font("TimesRoman", Font.BOLD, 18), 1);
```



## Defining automatic Grid redraw

```
{button Concepts,AL('Database Development  
Edition_Extension_Overview;Customizing_Database_Development_Edition_code',0,'')} {button See  
Also,AL('Grid_component;Adding_a_Grid_component_to_a_form_as_a_detail_view;Changing_Grid_Cell_Attributes;Protecting_a  
_Grid_column;Changing_Grid_column_attributes;Defining_automatic_Grid_row_numbering;Modifying_the_Grid_toolbar;Adding_  
Code_to_a_Java_Source_File_howto',0,)}}
```

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

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

Place any custom source code after the INIT\_CONTROLS code block.



## Defining automatic Grid row numbering

```
{button Concepts,AL('Database Development  
Edition_Extension_Overview;Customizing_Database_Development_Edition_code',0,'')} {button See  
Also,AL('Grid_component;Adding_a_Grid_component_to_a_form_as_a_detail_view;Changing_Grid_Cell_Attributes;Protecting_a  
_Grid_column;Changing_Grid_column_attributes;Defining_automatic_Grid_redraw;Modifying_the_Grid_toolbar;Adding_Code_to  
_a_Java_Source_File_howto',0,)}
```

Automatic row numbering can be turned on or off. When 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
```

Place any custom source code after the `INIT_CONTROLS` code block.

## Modifying the Grid toolbar

```
{button Concepts,AL('Database Development
Edition_Extension_Overview;Customizing_Database_Development_Edition_code',0,'')} {button See
Also,AL('Grid_component;Adding_a_Grid_component_to_a_form_as_a_detail_view;Changing_Grid_Cell_Attributes;Protecting_a
_Grid_column;Changing_Grid_column_attributes;Defining_automatic_Grid_row_numbering;Adding_Code_to_a_Java_Source_F
ile_howto',0,.)}
```

You can add additional 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

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.

Place any custom source code after the INIT\_CONTROLS code block.

### Adding search capability

The following example adds a few components to the toolbar and implements a standard search capability within a column. New features include a search field, a forward and backward search direction radio button, and a Go button (to start the search).

1. Create a basic project with a Grid.
2. When the Grid is functional and displays data, add import statements at the top of the Grid source code:

```
import symantec.itools.db.awt.Grid;
import symantec.itools.db.awt.Coordinate;
import symantec.itools.db.awt.DataNotAvailable;
import symantec.itools.db.awt.DefaultTvEventHandler;
```

These import statements provide the necessary classes to create the custom event handler.

3. Add the following code for a custom event handler. Code notes follow the code sample.

```
1: class MyEventHandler extends DefaultTvEventHandler {
2:     TextField tfSearch;
3:     Checkbox  cbNext, cbPrev;
4:     Button    btnGo;
5:     Grid      grid;
6:
7:     public void setupView (Grid gr) {
8:         super.setupView(gr);
9:         grid = gr;
10:
11:         gr.addToToolbar (tfSearch = new TextField(15));
12:         CheckboxGroup cbg = new CheckboxGroup();
13:         gr.addToToolbar (cbNext = new Checkbox("Next", cbg,
true));
14:         gr.addToToolbar (cbPrev = new Checkbox("Prev", cbg,
false));
15:         gr.addToToolbar (btnGo = new Button("Search"));
16:     }
17:
18:     public boolean handleToolbarEvent (Event e) {
19:         if (e.target == tfSearch && e.id == e.ACTION_EVENT) {
20:             findSubstring(tfSearch.getText(), 1,
cbNext.getState());
21:             return true;
22:         } else if (e.target == btnGo && e.id == e.ACTION_EVENT) {
23:             findSubstring(tfSearch.getText(), 1,
cbNext.getState());
24:             return true;
25:         }
26:         return super.handleToolbarEvent(e);
27:     }
28:
29:     void findSubstring(String query, int col, boolean forward) {
```

```

30:         if (col < 0 || col >= grid.getNumberOfCols()) {
31:             throw new IllegalArgumentException ("col="+col+1)+"
is not
32:             in view");
33:         }
34:         int start, stop = grid.getNumberOfVisibleRows() - 1;
35:         query = query.toUpperCase();
36:
37:         int step = forward ? 1 : -1;
38:         Coordinate curr = grid.getCurrentCellCoordinates();
39:         if (curr != null) {
40:             start = curr.row() + step;
41:         } else if (forward) {
42:             start = 0;
43:         } else {
44:             start = grid.getNumberOfVisibleRows() - 1;
45:             stop = 0;
46:         }
47:         int stringAt = -1;
48:         try {
49:             int index = -1;
50:             for (int row=start; row <=stop; row +=step) {
51:                 String currText = grid.getCellText(row+1,
52:                     col+1).toUpperCase();
53:                 index = currText.indexOf(query);
54:
55:                 if (index != -1) {
56:                     stringAt = row + 1;
57:                 }
58:             }
59:         } catch (DataNotAvailable ex) {
60:             ex.printStackTrace();
61:         }
62:
63:         if (stringAt != -1) {
64:             grid.setCurrentCell (stringAt, 2);
65:         } else {
66:             System.out.println("Did not find "+query);
67:         }
68:     }
69: }

```

## Code Notes

Lines 11 through 15 add the new components to the toolbar.

Lines 18 through 27 act as the new `handleEvent` method for the Grid. While most of the standard grid events are dealt with internally, the `handleEvent` method deals with a Return key action even from the Search TextField or a press of the Search Go button. In both cases, the `findSubString` method is being called on to perform the search.

Lines 29 through 68 make up the `findSubString` method. The arguments to `findSubString` are:

- the text to search for
- the column to search within
- the direction to search (forward or backward)

This method searches through the specified column to find a cell containing a string that matches the search string (after conversion to uppercase).

Lines 30 through 33 assure that the column to search is within the range of actual columns in the grid. If the column to search is not within the actual columns within the grid, the `IllegalArgumentException` is thrown.

Line 34 declares the start and stop position variables for the search based on the number of rows available to search.

Line 35 converts the query to uppercase for the search.

Line 37 determines the step for the search depending on the direction of the search. If the search is a forward search, the step is set to positive value of 1. If the search is a backwards search, the step is set to a value of negative 1.

Lines 38 through 46 determine the starting and stopping point of the search.

Lines 47 through 61 start to check each cell in the given column to see if the cell contains a match to the search string. If not, the next cell is selected and searched, and so on, until the search is successful or the search range has been exhausted.

Lines 62 through 67 either send a message to the console saying the search was unsuccessful or select the cell that contained the match.

### Plugging in the new capability

Once you have 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 here.

```
1:      . . .
2:      try {
3:          grid1 = new symantec.itools.db.awt.Grid(DBA_employee);
4:          grid1.addUndoButton();
5:          grid1.addDeleteButton();
6:          grid1.addUndeleteButton();
7:      } catch (symjava.sql.SQLException e) {
8:          System.out.println(e.getMessage());
9:          return;
10:     }
11:     grid1.reshape(24,12,528,327);
12:     keyPressManagerPanel1.add(grid1);
13:     //}}
14:     grid1.installEventHandler(new MyEventHandler());
15: }
```

### Code Notes

After adding the `Grid` to a project, lines 1 through 13 and 15 already exist. Line 14 was added to install the new `MyEventHandler` object to the grid component.

The line on which the `installEventHandler` method is placed is important. Notice that it is just below the line with the `//}}` tag. This tag declares the end of an area of code that is automatically maintained by Visual Cafe.

Any manual code placed inside the matching `//{{` and `//}}` tags may be removed by the Visual Cafe two-way development system when Visual Cafe does code maintenance. Placing the `installEventHandler` method outside the tags ensures that Visual Cafe does not remove or change your code.

## Protecting a Grid column

```
{button Concepts,AL('Database Development  
Edition_Extension_Overview;Customizing_Database_Development_Edition_code',0,'')} {button See  
Also,AL('Grid_component;Adding_a_Grid_component_to_a_form_as_a_detail_view;Changing_Grid_Cell_Attributes;Changing_  
Grid_column_attributes;Defining_automatic_Grid_row_numbering;Defining_automatic_Grid_redraw;Modifying_the_Grid_toolbar;  
Adding_Code_to_a_Java_Source_File_howto',0,,)}
```

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.

Following 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.

Obsolete



## Adding a Grid component to a form as a detail view

```
{button See Also,AL('Grid_component;dbAWARE_Join_property;dbAWARE_Binding_property',0,'','')}
```

You can define a [master/detail relationship](#) by adding a Grid component to a form as the detail view. Remember that you can get help on properties in the Property List by selecting a property and pressing F1. <<dbANYWHERE API only?>>

1. While the form is selected in the Project window, choose Insert ► Add Table Wizard.
2. Complete the wizard using only TextField components.  
Press F1 to get help on any wizard page.
3. In the Form Designer, delete all labels and fields that were automatically created for the table.
4. Drag a Grid component from the Component Library or Palette onto the form and size it appropriately.
6. Define the Grid's RelationView property. <<Doesn't work for JDBC API, right?>>



## Entering timestamp data

Timestamp data uses a 24-hour format.

If you enter

```
1997-02-05 11:00:00.000000000
```

then the value is interpreted by dbANYWHERE as 11:00am.

If you want the value of 1:00pm, enter

```
1997--02-05 13:00:00.000000000
```

Note that nine digits are used in the nanosecond position of the timestamp data, this is due to the java.sql.Timestamp class use of nine digits. Therefore, to enter 123 nanoseconds, you must pad the value with zeros. The value is:

```
000000123
```



## f1\_pro.doc

- *Contents*: This file contains the F1 context-sensitive help for Visual Cafe Database Development Edition .
- *Dialog box topics*: A wizard page topic can be accessed only from a wizard page or a Details button in a procedure topic (not from the index, no keywords) – because a wizard page topic doesn't make sense unless you are looking at the wizard page.

### History

1/16/97: Added topics for dbNAVIGATOR items to testing

2/12/97: Removed procedures and made them their own file.

3/10/97 -- Engineering review

8/27/97 – updated for vcafe Database Development Edition 2.0 (Visual Cafe dbDE)

Windows



## Server item

{button Concepts,AL('Database Development Edition\_Extension\_Overview;dbNAVIGATOR\_window',0,'')} {button See Also,AL('Connecting to a database;Connecting to a dbANYWHERE Server;dbNAVIGATOR pop-up menu',0,'')}

A dbANYWHERE Server item represents a dbANYWHERE Server that can be accessed by the computer on which Visual Cafe Database Development Edition is running.



## Data Source item

{button Concepts,AL('Database Development Edition\_Extension\_Overview;dbNAVIGATOR\_window',0,'')} {button See Also,AL('Connecting to a database;Connecting to a dbANYWHERE Server;dbNAVIGATOR pop-up menu',0,'')}

A Data Source item represents a database that is connected to the associated dbANYWHERE Server. When you connect to a dbANYWHERE Server, dbNAVIGATOR displays [data source](#) items for the databases that are connected to the dbANYWHERE Server.



## Database Table item

{button Concepts,AL('Database Development Edition\_Extension\_Overview;dbNAVIGATOR\_window',0,'')} {button See Also,AL('Connecting to a database;Connecting to a dbANYWHERE Server;dbNAVIGATOR pop-up menu',0,'')}

A Database Table item represents a table in the database. When you connect to a database, dbNAVIGATOR displays the tables and columns that are in the database.



## Database Column item

{button Concepts,AL('Database Development Edition\_Extension\_Overview;dbNAVIGATOR\_window',0,'')} {button See Also,AL('Connecting to a database;Connecting to a dbANYWHERE Server;dbNAVIGATOR pop-up menu',0,'')}

A Database Column item represents a column in the database table. When you connect to a database, dbNAVIGATOR displays the tables in a database and the columns in the tables.

## Dialog boxes



## Using the Join Definition dialog box and wizard page

```
{button Concepts,AL('Database Development Edition_Extension_Overview',0,'')} {button See  
Also,AL('Wizards_using;Creating_a_view;Changing_a_view;Component_Properties_overview;Changing_Component_Propertie  
s_howto;Property_Inspector_using_F1',0,'')}
```

### Property List ▶ Join property

The Join Definition dialog box lets you create or change a detail view's [join](#) that defines a [master/detail relationship](#). A master/detail join is useful when you have two or more tables that have a one-to-many relationship, such as a customer order that has multiple order items.

**Note** The Join Definition dialog overrides the SQL statement created for the detail [RelationViewPlus](#), depending on the Join criteria that is specified.

1. Choose a column from the master view by clicking a cell under Master Column and choosing a column from the list.
2. Choose a column from the detail view by clicking a cell under Detail Column and choosing a column from the list.
3. Choose an operator for the relationship between the two columns by clicking a cell under Operator and choosing an operator from the list.





## Using the Logon dialog box

{button Concepts,AL('Database Development Edition\_Extension\_Overview',0,'')} {button See  
Also,AL('Wizards\_using',0,'')}

The Logon dialog box lets you log on to a database.

1. Enter a user name.
2. Enter a password.
3. Select or clear the check box. If you select the check box, dbNAVIGATOR displays only the database tables owned by the specified user name.

## Using the SQL Statement dialog box

```
{button Concepts,AL('Database Development Edition_Extension_Overview',0,'')} {button See  
Also,AL('Specifying_SQL_SELECT_statements;Creating_a_view;Changing_a_view;Component_Properties_overview;Changing  
_Component_Properties_howto;Property_Inspector_using_F1',0,'')}
```

### Property List ▶ Select Statement property

The SQL Statement dialog box lets you change the Select Statement property of a single [RelationViewPlus](#). The Select Statement property is a [SQL statement](#) that is used to query your database(s).

**Note:** The values you enter in the Join Definition dialog box override the SQL statement created for the detail RelationViewPlus, depending on the Join criteria that is specified.

The SQL statement dialog box has the following sections:

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 RelationViewPlus. 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.

Wizard pages



## Using the Start wizard page

```
{button Concepts,AL('Wizards_using',0,'')} {button See  
Also,AL('Creating_a_dbAWARE_project;Adding_a_table_to_your_project;Creating_a_data_source_table_from_a_template',0,''  
'')}
```

The Start wizard page describes the wizard's main steps.



## Using the Project Type wizard page

```
{button Concepts,AL('Wizards_using',0,'')} {button See  
Also,AL('Creating_a_dbAWARE_project;Adding_a_table_to_your_project;Creating_a_data_source_table_from_a_template',0,'`  
'')}
```

The Project Type wizard page lets you choose the type of project to create.

Option	Description
Applet	A program that can be added to a Web page and run by Java-enabled Web browsers (or the AppletViewer).
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 can be used to show dialogs and windows.



## Using the dbANYWHERE Server dialog box and wizard page

```
{button Concepts,AL('Wizards_using;dbNAVIGATOR_window;Using_the_dbANYWHERE_application',0,'')} {button See  
Also,AL('Creating_a_dbAWARE_project;Adding_a_table_to_your_project;Creating_a_data_source_table_from_a_template',0,'`  
'')}
```

The dbANYWHERE Server dialog box lets you select a dbANYWHERE Server that is already connected or make a new dbANYWHERE Server connection. See the dbANYWHERE help (available from the help contents, for example) for more information on dbANYWHERE.

To access this dialog box, choose dbNAVIGATOR ► Insert Server pop-up menu option. It also appears when you first open dbNAVIGATOR, if you have not set up any server connections, and in the dbAWARE Project Wizard.

### To select a dbANYWHERE Server that is already connected

Choose a name from the dbANYWHERE Server Name list. The dialog box fills in the Host Name or IP Address and the Port Number for you.

### To make a new dbANYWHERE Server connection

1. Enter the name you want to call your dbANYWHERE Server while working in Visual Cafe.
2. Enter the dbANYWHERE Server's [host name or IP address](#).
3. Enter the dbANYWHERE Server's [port number](#).

### Troubleshooting

I can't connect to a server

- Do you have the right IP address and port number?
- Do you have access to the IP address?
- Is the target machine available? (try a "ping" utility)
- Is dbANYWHERE running on the target machine?
- Do you have the proper version of the dbANYWHERE class file?



## Using the Data Source wizard page

```
{button Concepts,AL('Wizards_using',0,'','')} {button See  
Also,AL('Creating_a_dbAWARE_project;Adding_a_table_to_your_project;Creating_a_data_source_table_from_a_template',0,'','  
'')}
```

The Database wizard page lets you select a database.

1. Choose a name from the database list. The list displays the databases that are connected to the dbANYWHERE Server you selected previously.
2. If you haven't connected to the database yet, the Database wizard page displays the Logon dialog box that lets you logon to the database.

► [Details on this step](#)



## Using the Database Table wizard page

```
{button Concepts,AL('Wizards_using',0,'','')} {button See  
Also,AL('Creating_a_dbAWARE_project;Adding_a_table_to_your_project;Creating_a_data_source_table_from_a_template',0,'','  
'')}
```

The Database Table wizard page lets you select a database table from the previously specified server.

Choose a name from the list.





## Using the Database Columns wizard page

```
{button Concepts,AL('Wizards_using',0,'','')} {button See  
Also,AL('Creating_a_dbAWARE_project;Adding_a_table_to_your_project;Creating_a_data_source_table_from_a_template',0,'','')}
```

The Database Columns wizard page lets you choose the database columns that you want to access. The list displays columns in the previously specified database.

To deselect a column, click its check box.



## Using the Components and Labels wizard page

```
{button Concepts,AL('Wizards_using',0,'')} {button See  
Also,AL('Creating_a_dbAWARE_project;Adding_a_table_to_your_project;Creating_a_data_source_table_from_a_template',0,'`  
')}
```

The Components and Labels wizard page lets you select a component and specify a label for each database column, or specify that the columns appear in a single grid.

### To choose a grid

1. Select Grid.
2. Type a grid label or clear the field.

### To choose individual components for each column

At runtime, the applet or application displays each column's data in its component and displays a label for the component. The wizard page displays a default component and label for each column.

1. Select Components.
2. To change a column's component, click the component name, then choose a component from the list that appears.
3. To change a column's label, click the text for the label, then type the new text..

**Note** If you don't want the component to have a label, click the text for the label and delete it.

**Tip** You can change the component's label after it is built by modifying the component's Text property.



## Using the Database Operations wizard page

```
{button Concepts,AL('Wizards_using',0,'')} {button See  
Also,AL('Creating_a_dbAWARE_project;Adding_a_table_to_your_project;Creating_a_data_source_table_from_a_template',0,'`  
')}
```

The Actions wizard page lets you select the [actions](#) that the applet or application provides. At runtime, the form displays a button for each action.

### To select database operations

Click the check box for each action you want enabled on the form through buttons.

### To change a button's text

1. Click the text.
2. Enter text for the button.

**Tip** To change properties of an action button after being generated by the wizard, you can use the Property List.

**Tip** To create an action button outside of the wizard, add a button component to the form. With the button component selected, choose the Add Interaction option from the right-mouse pop-up menu. Use the Interaction Wizard to specify the appropriate action for the button.



## Using the Finish wizard page

```
{button Concepts,AL('Wizards_using',0,'')} {button See  
Also,AL('Creating_a_dbAWARE_project;Adding_a_table_to_your_project;Creating_a_data_source_table_from_a_template',0,'`  
')}
```

The Finish wizard page displays the choices you made during the wizard.

To change a selection, go back to the wizard page where you made that selection by using the Back button.



## Using the Master View wizard page

```
{button Concepts,AL('Wizards_using',0,'')} {button See  
Also,AL('Creating_a_dbAWARE_project;Adding_a_table_to_your_project;Creating_a_data_source_table_from_a_template',0,'`  
')}
```

The Master View wizard page lets you select a table for a master [view](#) in a [master/detail relationship](#).

The table you choose in this wizard page is the master view and the table you are creating is the detail view.

The list box displays the tables that represent [QueryNavigator](#) or [RelationViewPlus](#) components that exist in your project.



## Using the dbAWARE Template wizard page

```
{button Concepts,AL('Wizards_using',0,'','')} {button See  
Also,AL('Creating_a_dbAWARE_project;Adding_a_table_to_your_project;Creating_a_data_source_table_from_a_template',0,'','  
'')}
```

The Template wizard page lets you choose a template for your database table.

Visual Cafe Database Development Edition provides the template set.

This file contains the error message topic

2/12/97: Created and added to project, new for release 1.0d



## Visual Cafe Database Development Edition error messages

### **symantec.itools.db.net.NetException: Server not responding**

#### What does this mean?

The Java client was not able to establish a connection with the specified dbANYWHERE server.

#### What do I do about it?

Make sure dbANYWHERE is running normally at the location specified in the connection URL. If the server is remote, make sure there is a good network connection to it.

### **Client/server mismatch**

#### What does this mean?

Your version of dbANYWHERE does not match your version of Visual Cafe Database Development Edition.

#### What do I do about it?

You need a new version of dbANYWHERE. Matching versions of dbANYWHERE and Visual Cafe Database Development Edition ship together.



# Glossary

This is the glossary for Visual Cafe, Visual Cafe Database Development Edition, and dbANYWHERE:

- Definitions
- URLs (not pop-ups because they're too large)
- dbawdsn.ini fields (for dbANYWHERE)
- Fields in Data Source configuration files (for dbANYWHERE)

## How it was created

1. Started with defin.doc (the glossary for Visual Cafe).
2. Put entries in alphabetical order.
3. Added definitions from the Pro glossary (progloss.doc) and the dbaw glossary (db\_pop.doc).
4. Verified that all definitions in the Visual Cafe User's Guide glossary (defin.doc) are already in here.
5. Created and attached glossary.dot for the styles (glossary.dot is a revised version of robotf.doc).
6. Combined duplicate entries.
7. Modified descriptions (so that they are clearer).
8. Added descriptions: JDK, Java API, Hierarchy Editor, dbNAVIGATOR, Project window, Form Designer, server, foreign key, optimistic concurrency, primary key, stored procedure.
9. Create the main glossary topic which can be added to the .cnt files. For the bitmaps, I used the samples provided on the CD that came with *Developing Online Help* by Boggan, Farkas, and Welinske. You may want to use different bitmaps because I don't know if it's legal to use these and because someone can probably create more attractive ones.

## Ramifications

1. Files for projects: Each help project needs to:
  - Delete the current glossary/definitions file (defin.doc, progloss.doc, or db\_pop.doc).
  - Add this file (gloss.doc).
  - Make whatever modifications are necessary to access the definitions in this file.
2. Duplicate entries:
  - debug mode: Kept the Debug\_mode\_glossary topic ID. Deleted the Break\_mode\_glossary topic ID.
  - Java Virtual Machine: Kept the Java\_Virtual\_Machine\_glossary topic ID. Deleted the Java\_VM\_Java\_Virtual\_Machine topic ID.
  - database view, global view, and two views: Deleted database view and global view (this information was in the other view definitions). Combined the two view definitions. Kept the View\_glossary topic ID. Deleted the global\_view\_glossary and view topic IDs (there were two of these).
  - URLs: Deleted the URL entry because it has been superseded by the dbANYWHERE Server machine URL entry. Deleted the URL topic ID.
  - class method, instance method, method: Deleted the class method and instance method entries. Kept the Method\_glossary topic ID. Deleted the Class\_method\_glossary and Instance\_method\_glossary topic IDs.
  - component and visual object: Deleted the visual object entry and its Visual\_Object\_glossary topic ID. Kept the Component\_glossary topic ID.
3. Changed entry names:
  - Changed class variable and instance variable to variable. Kept the Class\_variable\_glossary topic ID. Deleted the instance\_variable topic ID.
  - Changed class member to member. Kept the same topic ID.
  - Changed invisible component and non-visible object to hidden component. Kept the Invisible\_object\_glossary topic ID. Deleted the Non\_visual\_object\_glossary topic ID.
4. Deleted entries:

- control (topic ID = Control\_glossary).
5. Since the main glossary topic uses mid-topic jumps, you must redefine the visual window so that it does not auto-size the height.

**To do**

1. action: Make sure all the valid actions are listed.
2. Data Source: what spreadsheets are supported? should this information go in the glossary definition or in the "Installing dbANYWHERE Server" manual?

Main glossary topic

## Glossary

a

b

c

d

e

f

g

h

i

i

k

l

m

n

o

p

q

r

s

t

u

v

w

x

y

z

### A

[abstract class](#)

[access type](#)

[action](#)

[active window](#)

[adapter](#)

[anchor component](#)

[applet](#)

[applet tag](#)

[application](#)

## **B**

[bean](#)

[boolean expression](#)

[Breakpoints window](#)

[bytecode](#)

## **C**

[call stack](#)

[Callswindow](#)

[class](#)

[Class Browser](#)

[client machine](#)

[client software](#)

[component](#)

[Component Library](#)

[Component Palette](#)

[conditional breakpoint](#)

[connection statistics](#)

[ConnectionInfo](#)

[ConnectionManager](#)

[container](#)

[context menu](#)

[custom code](#)

## **D**

[data binding](#)

[data source](#)

[database template](#)

[dbANYWHERE](#)

[dbANYWHERE API](#)

[dbAW subprotocol](#)

[dbAWARE component](#)

[dbNAVIGATOR](#)

[debug mode](#)

[Debug toolbar](#)

[Debugging window](#)

[declaration](#)

[definition](#)

[design time](#)

[development machine](#)

[dialog box](#)

[DML](#)

## **E**

[event](#)

[event adapter](#)

[event binding](#)

[event handler](#)

[event listener](#)

[exception](#)

## **F**

[foreign key](#)

[form](#)

[Form Designer](#)

[frame](#)

## **G**

[garbage collection](#)

## **H**

[Hierarchy Editor](#)

## **I**

[identifier](#)

[incremental debugging](#)

[inheritance](#)

[inner class](#)

[instance](#)

[interaction](#)

[interface](#)

[introspection](#)

[invisible component](#)

## **J**

[Java API](#)

[Java file](#)

[Java Archive \(JAR\) file](#)

[Java Virtual Machine](#)

[JavaBeans](#)

[JavaBeans component](#)

[JDBC API](#)

[JDBC-ODBC bridge](#)

[JdbcConnection](#)

[JDK: Java Developer's Kit](#)

[join](#)

## **K**

(none)

## **L**

[layout](#)

[list binding](#)

[listener](#)

## **M**

[macro](#)

[manifest file](#)

[master/detail relationship](#)

[mediator](#)

[member](#)

[method](#)

[Messages window](#)

[modal](#)

[modeless](#)

## **N**

[nested class](#)

## **O**

[ODBC](#)

[ODBC subprotocol](#)

[object](#)

[Objects view](#)

[optimistic concurrency](#)

[overloading](#)

[overriding](#)

## **P**

[package](#)

[Packages view](#)

[panel](#)

[ping: Packet INternet Groper](#)

[primary key](#)

[project](#)

[Project template](#)

[Project window](#)

[projection binding](#)

[property](#)

[Property List](#)

## **Q**

[QueryNavigator](#)

## **R**

[RecordDefinition](#)

[refresh](#)

[result set](#)

[regular expression syntax](#)

[RelationViewPlus](#)

[run mode](#)

[run time](#)

## **S**

[scope](#)

[Session](#)

[server](#)

[Source pane](#)

[Source window](#)

[SQL](#)

[SQL statement](#)

[stored procedure](#)

## **T**

[thread](#)

[Threads window](#)

[top-level component](#)

[transaction isolation levels](#)

## **U**

(none)

## **V**

[ValueTip](#)

[variable](#)

[Variables window](#)

[view](#)

[Visual Cafe window](#)

## **W**

[Watch window](#)

[window](#)

[workspace](#)

## **X**

(none)

## **Y**

(none)

## **Z**

(none)



## Definitions

A

**abstract class**

A class that contains an abstract method and is incompletely defined; that is, at least one method is not complete. You cannot instantiate an abstract class.

## **access type**

Scope of a variable or method:

- **Public:** Accessible from any class.
- **Protected:** Accessible from the class that defines the variable or method and from the class' subclasses.
- **Private:** Accessible from the class that defines the variable or method.
- **Package:** Accessible from any class in the package that defines the variable or method. This is the default access type.

## **action**

Action performed on a database:

- delete: Deletes the current record from the Data Source.
- 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.

**active window**

Window which receives keyboard input. Only one window can be active at any time.

## **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.

## **anchor component**

Component which acts as the reference point when modifying a group of components. When you select multiple components, the last component selected is the anchor component. It has distinct, colored selection handles.



## **applet**

### **Program**

Special type of Java program that you can add to a Web page. To run an applet:

- Add the applet to a Web page and display the Web page in a Java-enabled Web browser.

-or-

- Run the applet from the Applet Viewer. (The Applet Viewer is a tool that Sun provides with the JDK. The Applet Viewer is also included in Visual Cafe and Visual Cafe Database Development Edition.)

### **Class**

Class in the Java API.

## **applet tag**

An applet tag is HTML code that causes an applet to appear in a Web page. It has the following basic format:

```
<APPLET code="applet.class" width=pixw height=pixh></APPLET>
```

*applet* is the name of the applet.

*pixw* is the number of pixels for the width.

*pixh* is the number of pixels for the height.

Consult an HTML book for more information on the applet tag.

## **application**

### **Program**

Java program you can run from a computer that has the Java Virtual Machine.

### **Template**

Template provided by Visual Cafe (and Visual Cafe Database Development Edition) when you choose File ► New Project ► Basic Application. This template creates a class that is an extension of the Frame class. This class is a good base for a main application window.

B

## **bean**

A component complying with the JavaBeans standard. Also called a JavaBean or a JavaBeans component. A bean is a reusable component that can be visually manipulated in a builder tool. 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.

## **Breakpoints window**

Window that contains a list of all breakpoints in a project. Use this window to add, remove, or modify breakpoints. You can set simple breakpoints that stop execution at a certain line or method, or conditional breakpoints based on an expression.

To see the Breakpoints window, choose View (or Window) ► Breakpoints.

**bytecode**

Machine-independent code generated by the Java compiler and executed by the Java interpreter.



C

## **call stack**

An area reserved in memory by the compiler to keep track of all method calls that are made.

## **Calls window**

Debugger window that shows all the active method calls leading to the current process. When an applet or application calls a method, the Java Virtual Machine (Java VM) adds the method to the stack. When the method returns, the Java VM takes it off the stack. The currently executing method is on the top of the stack and the previous methods called are below it. This window is useful for following the flow of your code, for example.

To see the Calls window, View (or Window) ► Call Stack.

## **class**

Collection of variables and methods that you can use to define an object. The variables define the class structure. The methods define the class behaviors.

When compiled as a bytecode program (versus a native Win32 executable), a Java source file becomes one or more class files.

## Class Browser

A three-pane window that lists the Java classes in your project and the methods and data members contained within each class:

- Classes pane — classes
- Members pane — methods and data in the selected class
- Source pane — source code of the selected member

Both the Classes and Members panes support keyboard incremental searches. As you type the name of a class or member, the list of matching items is refined until the class or member you want is automatically selected. You can display classes and members in a variety of views and filter the items that are displayed. The Source pane provides the same editing features as the Source window, while ensuring that you do not unintentionally change code outside of the object's scope.

To see the Class Browser, choose View (or Window) ► Class Browser.

**client machine**

Machine that communicates with a dbANYWHERE Server machine to access a Data Source.

## **client software**

Software that dbANYWHERE uses to communicate with a database engine such as:

- Oracle Server client software
- Microsoft SQL Server client software
- ODBC32 Administrator

## component

A JavaBeans component.

- A *visual component* is a user interface element, such as a window, menu, button, and so on, that is visible at run time and appears in the Visual Cafe Project window and Form Designer. It extends from the Java Component class and has a screen position, a size, and a foreground and background color.
- An *invisible component* is not visible at run time, such as a Timer, or displays in a different way in the Form Designer and at run time, such as a MenuBar. It does not extend from the Java Component class. In the Form Designer, an invisible component is represented by an icon that does not effect the form layout.
- Some components can contain other components, such as an application window containing a button; these components are called *containers*. In the Objects view of the Project window, the components in a container appear subordinate to the container, like a file system display. The containers at the top level are separate Java files in your project (called Visual Cafe *forms*), while the components in the containers are Java code within the container Java file.
- To create your user interface, you can display a form in the Form Designer, then drag onto the form a variety of components from the Component Library or Palette; you assign the component properties in a separate Property List window, and add interactions between components with the Interaction Wizard. Visual Cafe automatically creates the Java code for you during this design process.
- Components are like controls in C++.



## Component Library

Collection of components that you can add to your applet or application. When you create a project template or add a component to the Component Palette, Visual Cafe adds the component to the Component Library. To see the Component Library:

View (or Window) ► Component Library

## **Component Palette**

Palette that provides easy access to components. You can place any component in the Component Library on the Component Palette.

**conditional breakpoint**

Breakpoint that lets you stop program execution when a specified expression evaluates to true.

## **connection statistics**

DC messages which occur when a client disconnects. The format for a DC message is:

DC, <client IP address>, <initial connect time>, <connect time duration>, <number of requests processed>

where:

- <initial connect time> is in second since 1970
- <connect time duration> is in seconds

## ConnectionInfo

For the dbANYWHERE API, when you use a database wizard or drag a Data Source item to a project, Visual Cafe Database Development Edition creates a ConnectionInfo object, showing the database you are connected to. It includes the default user name and password, which you can remove. The object supports multiple queries and updates simultaneously, reducing database resource requirements.

## **ConnectionManager**

A bean, used with the JDBC API, as a logical container for JdbcConnection beans. Dragging a Data Source item from dbNAVIGATOR to a project creates ConnectionManager and JdbcConnection objects, which manage connections to a data source. You have one ConnectionManager object per project, and one JdbcConnection object per data source. You can also drag ConnectionManager objects to the Component Library.

## **container**

Component that can contain other components. For example, Applet and Panel are containers.

A top-level container, also called a Visual Cafe form, is at the top level in the Objects view of the Project window. It has a corresponding Java file that appears in the Packages and Files views.

**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 data source. The data binding is the same whether your data is being obtained from the JDBC API or the dbANYWHERE API. The components adhere to a standard interface for the binding. If the component is not aware of the data-binding interfaces, the binding can be controlled by a mediator bean.

## **data source**

Contains the information for creating a database connection, like the name of the database, its server, and its network location. A data source 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 data source. Sometimes a data source is called a DSN (Data Source Name). For information about the Data Sources dbANYWHERE supports, see the dbANYWHERE manual.

## **database template**

Forms which are pre-designed. You can choose amongst these by using the dbAWARE Template Wizard. Each Visual Cafe database template is designed for a different kind of information.

## **dbANYWHERE (naming conventions)**

dbANYWHERE Server	Middleware database software that supports three-tier architecture for database access. Symantec dbANYWHERE is a database connectivity server that provides an implementation of the JavaSoft JDBC API as well as its own powerful java database connectivity API, called the dbANYWHERE API.
dbANYWHERE Workgroup Server	Version of the dbANYWHERE Server which is shipped with Visual Cafe Database Development Edition that restricts the number of licenses for use to five.
dbANYWHERE Server machine	Hardware on which dbANYWHERE software is running
dbANYWHERE	Both versions of dbANYWHERE

## **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. Symantec dbANYWHERE is a database connectivity server that provides an implementation of the JavaSoft JDBC API as well as its own powerful Java database connectivity API, called the dbANYWHERE API.

## **dbAW subprotocol**

A JDBC subprotocol supported by Symantec dbANYWHERE, which is a JDBC driver. Another JDBC subprotocol supported by Visual Cafe Database Development Edition is ODBC, which is used by the free JDBC-ODBC bridge (another JDBC driver). The JDBC API requires a URL to establish a connection to a database; the subprotocol is part of this URL syntax:

***jdbc:dbaw://host-name-or-IP-address:port-number/data-source-name***

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.



## **dbAWARE component**

Visual Cafe component that has additional properties for binding the component to a database row, column, table, or result set.

## **dbNAVIGATOR**

Browser window which shows the servers, data sources, and contents of the data sources that are connected to those servers. You can also use the dbNAVIGATOR perform drag-and-drop operations on your forms and components. To see the dbNAVIGATOR:

View (or Window) ► dbNAVIGATOR

## **debug mode**

Temporary suspension of a running program. In debug mode, you can edit code, edit forms, set or clear breakpoints, debug threads, and view the calls on the call stack. After making any edits, restart program execution to see the effect of the changes.

## **Debug toolbar**

Toolbar that has buttons for debugging.

## **Debugging window**

Window for debugging an applet or application. The Debugging window is a special mode of the Source window: It is the Source window while you are debugging.

## **declaration**

Statement that establishes an identifier and associates attributes with it, without necessarily reserving its storage for data or providing the implementation for methods.

**definition**

Declaration that reserves storage for data or provides implementation for methods.

**design time**

Time during which you build an applet or application in the development environment.



**development machine**

Machine that combines a client to a database server and a Java development environment such as Visual Cafe Database Development Edition.

## **dialog box**

Window with a title bar that can receive and process input from a user.

- Applets cannot use dialog boxes.
- Dialog boxes are not suitable for an application's main window.
- Dialog boxes can contain components, but not menus.
- Use dialog boxes for temporary windows.

## **DML**

Data Modification Language, a subset of SQL.

E

**event**

Action or occurrence to which an object can respond. Events are typically user actions that the program can capture and respond to. For example, mouse clicks, key presses, and mouse movements are events.

## **event adapter**

Event listener interface that is implemented as a class.

## **event binding**

Code that enables an object to receive and process an event. It is made of three parts: the event handler, the listener or adapter implementation, and the code to register the listener or adapter to the object triggering the event.

## **event handler**

A method that is called when a certain type of event is triggered. Visual Cafe automatically generates the code needed to bind the occurrence of an event to an event handler when you create an interaction with the Interaction Wizard or from the Events/Methods pull-down menu of the Source window. An event binding is made of three parts: the event handler, the listener or adapter implementation, and the code to register the listener or adapter to the object triggering the event. The default name of an event handler is the object name, followed by an underscore then the name of the action that triggers the event.



## **event listener**

An object that has defined the listener interface for a specific event. After this interface has been implemented in a class, an instance of this class may be registered as an event listener. When an event is generated, the event is sent to the object, as well as all other registered listeners.

**exception**

An event that occurs during the execution of your program that interferes with, disrupts, or stops the normal flow of instructions.

F

**foreign key**

Column or columns in one 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.

## **form**

A component that can only appear at the top level in the Objects view of the Project window. Applet, Frame, Window, and some dialog components are forms. In the Component Library, the Visual Cafe forms are all in the Forms group. When you open a form in the Form Designer, the form boundaries are the bounds of the window. You can position other components, such as text, buttons, and graphics, on the form; these components are contained by the form.

## **Form Designer**

Window that visually displays form components so you can design the user interface of the form.

**frame**

Window which can contain components and menus.

G



## **garbage collection**

Automatic detection and freeing of memory that is no longer in use. The Java run time system performs garbage collection so you don't have to explicitly free the memory associated with objects and other data.

H

## **Hierarchy Editor**

Window that displays the class hierarchy for your applet or application. To see the Hierarchy Editor:

View (or Window) ► Hierarchy Editor



**identifier**

Name of an item in a Java program.

## **incremental debugging**

A feature that enables you to edit code while your program is executing or paused in the Visual Cafe debugger. Also called run-time editing. You enable this feature by choosing Tools ► Environment Options

► Debugging.

## **inheritance**

Concept wherein classes automatically contain the variables and methods defined in their superclasses.

## **inner class**

A class that is included within the body of another class, even within a method (called a local class). An inner class is also called a nested class. This feature is new for JDK 1.1 and is useful for creating adapter classes. After compilation, the inner class ends up in its own class file, which has a dollar sign (\$) in its name.



## **instance**

Data item based on a class. An instance of a class is usually called an object. For example, multiple instances of a Form class share the same code and are loaded with the same components with which the Form class was designed. At run time, the individual properties of components on each instance can be set to different values.

## **interaction**

Relationship between two or more components, or a component and itself. The components may be on the same form or on different forms. An interaction consists of:

- One or more components (trigger component and action component)
- Trigger event
- Action

For example, you can connect a button (the trigger component) to a text box (the action component) so that when the user clicks on the button (the trigger event), the associated text box is enabled for user input (the action).

## **interface**

A set of methods and constants to be implemented by another object. It defines the behavior, or certain characteristics, that another object implements. An interface can define abstract methods and final fields, but not the implementation of them.

## **introspection**

The ability to read JavaBeans classes directly with the Core Reflection API using the Introspector class. This information is stored in a BeanInfo object and includes data such as properties, events, and all the accessible methods.

## **invisible component**

Component that is not visible at run time, such as a Timer, or displays in a different way in the Form Designer and at run time, such as a MenuBar. It does not extend from the Java Component class. In the Form Designer, an invisible component is represented by an icon that does not effect the form layout. To toggle between displaying and hiding invisible components at design time:

Layout ► Invisibles

J

## **Java API: Java Application Program Interface**

API provided by the JDK. For more information, see the Java Web page ([java.sun.com](http://java.sun.com)).

**Java file**

File that contains components and Java source code.



## **Java Archive (JAR) file**

Compressed archive file that complies with the JavaBeans standard. It is the primary method for delivering JavaBeans components. For example, a JAR file can contain one or more related beans, and any support files, including classes, icons, graphics, sounds, HTML documentation, serialization files, and internationalization files. You can deploy applets and applications from a JAR file. A JAR tool, called `jar.exe` on Windows computers, archives and extracts JAR files and is provided with JDK 1.1. In Visual Cafe, to use the JavaBeans components in a JAR file, you must first add the file to the Component Library.

## **Java VM: Java Virtual Machine**

Virtual machine provided with the JDK (Java Development Kit). The machine contains:

- Bytecode translator that converts a downloaded binary Java file into instructions that the client machine can execute.
- Library routines that a Java applet calls.

For more information, see the Java Web page ([java.sun.com](http://java.sun.com)).

## **JavaBeans**

A standard for creating portable, cross-platform components.

## **JavaBeans component**

A component complying with the JavaBeans standard. Also called a bean or JavaBean. A bean is a reusable component that can be visually manipulated in a builder tool. 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.

## **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

A free JDBC driver. 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*

**jdbc:dbaw:***//host-name-or-IP-address:port-number/data-source-name*

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 Data Source item from dbNAVIGATOR to a project creates ConnectionManager and JdbcConnection objects, which manage connections to a data source. You have one ConnectionManager object per project, and one JdbcConnection object per data source. You can drag either of these objects to the Component Library.

**JDK: Java Developer's Kit**

Tools for developing Java applets and applications. The JDK is included in Visual Cafe and Visual Cafe Database Development Edition. It is also available on the Java Web page ([java.sun.com](http://java.sun.com)).



## **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.

K

L

## **layout**

Property for container objects like forms and panels. The Layout property automatically arranges components in a container so that they display well on different platforms, Web browsers, screen sizes, and resolutions.

## **list binding**

Data binding that lets you display data from multiple database columns. List binding is helpful for populating list boxes.

## **local variable**

Data item defined within a block of code and accessible only by the code within the block. For example, a variable defined in a Java method is a local variable and can't be used outside the method.

M

## **macro**

Sequence of keystrokes. The Source editor's macro facilities let you record, save, and edit macros.



## manifest file

A file that describes the contents of an archive, such as a JAR. The file provides information on certain parts of the archive and can provide information about any JavaBeans in a JAR. It is made of sections separated by empty lines. Each section has one or more headers. Each header uses a *attribute:value* format where *attribute* specifies various attributes of the archive contents and *value* is the relative name of the file being described. Attributes used include Manifest-Version, Name, Java-Bean, Digest-Algorithms, and XXX-Digest (related to Digest-Algorithms). See Sun Microsystem's Manifest File specification for more information.

## **master/detail relationship**

A one to many relationship between two database tables. The join property of the detail QueryNavigator or RelationViewPlus component defines the column of the master view which is the common attribute in the one to many relationship between the tables. For instance, you might want to generate a result set which enumerates all customers who purchased a product called the Spectacular Widget. You create this master/detail relationship by joining the Spectacular Widget column of a Sales table (which also has columns for numbers of sales and price) in an SQL query that returns a list of all the customers in a Customer database table who purchased a Spectacular Widget.

## **mediator**

Bean that lets you make a component database-aware. It is a bridge between the component and the QueryNavigator or RelationViewPlus components.

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 Java code, you can add a mediator to the component code, thereby encompassing its functionality within your component.

## **member**

Variable or method defined in a class.

## **method**

Behavior that acts on an object. A method is defined in a class.

- Class method: Method invoked using a class name.
- Instance method: Method invoked using the name of an instance (object).

## Messages window

Window that displays all Visual Cafe informational and error messages. For example, if Visual Cafe detects an error during parsing, the error message is displayed here. You can double-click an error message to open the file in which the error was detected. To see the Messages window:

View (or Window) ► Messages

## **modal**

Window or dialog box behavior that requires you to take some action before the focus can switch to another window or dialog box.

**modeless**

Window or dialog box behavior that does not require you to take some action before the focus can switch to another window or dialog box.



N

## **nested class**

A class that is included within the body of another class, even within a method (called a local class). A nested class is also called an inner class. This feature is new for JDK 1.1 and is useful for creating adapter classes.

O

## **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 dbANYWHERE server also communicates to other major database types through direct native calls to the client APIs (application programming interfaces) for those databases.

## ODBC subprotocol

A JDBC subprotocol supported by the JDBC-ODBC bridge, which is a free JDBC driver. (Each JDBC driver supports certain subprotocols.) Another subprotocol supported by Visual Cafe Database Development Edition is dbAW, which is used by Symantec dbANYWHERE. The JDBC API requires a URL to establish a connection to a database; the subprotocol is part of this URL syntax:

**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.

## **object**

Instance of a class.

## Objects view

View of a project that displays only visible components. To toggle between displaying and hiding hidden components at design time:

Layout ► Invisibles

**optimistic concurrency**

Locking technique that maximizes concurrency. The database management system (DBMS) locks data that is being modified but leaves alone data that is being viewed.



## **overloading**

Using one identifier to refer to multiple items in the same scope. In Java, you can overload methods but not variables or operators.

## **overriding**

Providing a different implementation of a method in a subclass of the class that originally defined the method. Overridden methods have the same name but take different parameters.

P

## **package**

Group of classes and interfaces.

## Packages view

View of a project that displays both visible and hidden components. To toggle between displaying and hiding hidden components at design time:

Layout ► Invisibles

## **panel**

Container that you can add to another container. A panel doesn't have a visible border. You can use a panel to group a window into logical regions.

**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 or columns in a database table that uniquely identify a record. A value in a primary key column cannot be NULL and must be unique.



**project**

Set of components and code that comprise an applet or application.

**project template**

Collection of components that you can use as the foundation for an applet or application. When you choose a template for a new project, the new project inherits all of the template's components. For example, you can create project templates for Web sites, single documents, and applets.

## **Project window**

Window that displays a project's objects or packages depending on which tab you select.

## **projection binding**

Data binding to one row in a database column.

## **property**

Attribute of a component. Properties define component characteristics such as size, color, label, or the state of an object, such as enabled or disabled. The Property List displays the properties of a project's components. To see the Property List:

View (or Window) ► Property List

## Property List

Window that displays a component's properties. To see the Property List:

View (or Window) ► Property List

Q

## QueryNavigator

A bean, used with the JDBC API, that manages a set of records. Dragging a Data Table item from dbNAVIGATOR to a form in a project creates a top-level RecordDefinition object and a QueryNavigator object, which is contained by the form. You can think of the RecordDefinition component as the data, and the QueryNavigator component as a way of looking at the data. After you have a QueryNavigator and RecordDefinition component in a project, you can create a master/detail relationship. You can drag a QueryNavigator object to the Component Library.



R

## **RecordDefinition**

A bean, used with the JDBC API, that is used 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 QueryNavigator object, which is contained by the form.

**refresh**

Updates the dbNAVIGATOR window to reflect changes made to database tables and columns it is displaying.

## regular expression syntax

Syntax that lets you perform regular expression matching. Regular expressions are wildcard characters. The pattern you search for can be a text string or a regular expression.

### Wildcard characters

?	Any character.
*	Zero or more occurrences of any character.
@	Zero or more occurrences of the previous character or expression.
% or <	Beginning of a line.
\$ or >	End of a line.
[...]	Any of the characters listed between [ and ]. You can use a hyphen (-) to specify a range of characters. For example, [abc] matches a, b, or c; [a-z] matches any lowercase letter; [A-Za-z] matches any upper or lowercase letter.
[~...]	Any character except those listed between [~ and ]. You can use a hyphen (-) to specify a range of characters. For example, [~A] matches any character but A; [~abc] matches any character except a, b, or c; [~A-Za-z] matches any non-alphabetic character.
\	Take the following character literally instead of using it as a wild card character. For example, you can use \* to search for an asterisk or \\ to search for a backslash character.
\t	Tab character.
\f	Formfeed character.

## **RelationViewPlus**

Database-aware 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 Development 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.

## **run mode**

Mode wherein your applet or application is running. In run mode:

- You can interact with your program as a user.
- Visual Cafe replaces the Project Menu with the Debug Menu.

## **run time**

Time when your applet or application is running. At run time:

- You can interact with your program as a user.
- You can pause the application and start debugging by pressing `CONTROL+BREAK`.



S

## **scope**

Access type of a variable or method:

- **Public:** Accessible from any class.
- **Protected:** Accessible from the class that defines the variable or method and from the class' subclasses.
- **Private:** Accessible from the class that defines the variable or method.
- **Package:** Accessible from any class in the package that defines the variable or method. This is the default access type.

## **Session**

For the dbANYWHERE API, when you use the database wizard or drag a dbANYWHERE Server item to a project, Visual Cafe Database Development 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.

## **Source pane**

Pane in the Source window and Class Browser that lets you view and edit code.

## Source window

Window that displays and lets you edit a Java source file, an HTML file, or a text file. The Source window is also available when you are debugging. To see the Source window:

- Select a file in the Project window, then choose Object ► Edit Source, or right-click and choose Edit Source.
- Double-click a file in the Packages or Files view of the Project window.
- Double-click in the Form Designer.
- Open a file by choosing File ► Open.

## **SQL**

Standard query language which is used for retrieving information from a relational database. The acronym stands for Structured Query Language.

## **SQL statement**

SQL string used to query a database and thereby define a view. If the string is in the standard SELECT statement format as shown below, the Visual Cafe Database wizards and Join Definition dialog box 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 and triggers can include control statements that allow repetition (LOOP\_statement) and conditional execution (IF\_statement and CASE\_statement) of SQL statements.

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.

T

**thread**

Path of execution in a running application. For example, an application can have threads that handle background processes that aren't visible to the user.

## Threads window

Debugger window that displays all the existing threads your program has created and the state of each thread. You can pause individual threads, which causes their execution to cease temporarily while all other threads continue to execute, and resume them. This helps you check for and resolve thread synchronization errors, where more than one thread is in contention for the execution of a method. Double-clicking a thread in this window updates the Calls window, so it reflects the scoping level of the thread.

To see the Threads window, choose View (or Window) ► Threads.

## **top-level component**

A component that can only appear 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.

## transaction isolation levels

The *isolation level* defines degree to which the operations in one transaction are visible to the operations in a concurrent transaction. Isolation levels prevent some or all inconsistent behavior and can be different for each connection. All isolation levels guarantee that each transaction will execute completely or not at all, and that no updates will be lost. This ensures recoverability at all times, regardless of the isolation level.

There are three types of inconsistency that can occur during the execution of concurrent transactions:

**Dirty read** — Transaction A modifies a row. Transaction B then reads that row before transaction A performs a COMMIT. If transaction A then performs a ROLLBACK, transaction B will have read a row that was never committed.

**Non-repeatable read** — Transaction A reads a row. Transaction B then modifies or deletes the row and performs a COMMIT. If transaction A then attempts to read the same row again, the row will have been changed or deleted.

**Phantom row** — Transaction A reads a set of rows that satisfy some condition. Transaction B then executes an INSERT, or an UPDATE (that generates one or more rows that satisfy the condition used by transaction A) and then performs a COMMIT. Transaction A then repeats the initial read and obtains a different set of rows.

Here are the types of transaction isolation levels:

TRANSACTION\_NONE — Transactions are not supported.

TRANSACTION\_READ\_UNCOMMITTED — Lets dirty reads, non-repeatable reads, and phantom reads occur.

TRANSACTION\_READ\_COMMITTED — Prevents dirty reads, but not non-repeatable reads and phantom reads.

TRANSACTION\_REPEATABLE\_READ — Prevents dirty reads and non-repeatable reads, but not phantom reads.

TRANSACTION\_SERIALIZABLE — Prevents dirty reads, non-repeatable reads and phantom reads.

U

V



## **ValueTip**

Interface element in Visual Cafe that displays the value of the variable that is under the cursor in the Source window in debug mode.

## **variable**

Structure in memory which holds data that has been assigned to it. A variable that is defined in a class defines the class' structure.

- Class variable: Variable associated with a class and not with a particular instance of the class.
- Instance variable: Variable associated with an instance of a class (an object).

## **Variables window**

Debugger window that shows the variables that are active in the current context. To change a variable value at run time, edit its value in this window.

To see the Variables window, choose View (or Window) ► Variables.

## **view**

### **In a development environment**

Window that shows a particular type of information.

- Global view: View that is accessible from the View (or Window) menu. The global views are: Breakpoints window, Calls window, Class Browser, Component Library, Component Palette, Hierarchy Editor, Output window, Property List, Threads window, Variables window, Watch window.
- Local view: View that is local to a project. The local views are: Project window, Source window.

### **For Data Sources**

A QueryNavigator and a RecordDefinition component, or a RelationViewPlus component, represent a view. A view has three parts:

- It is a collection or set of objects that contain data. You can set focus to one member of the set at a time by scrolling forward and backward through the set.
  - A QueryNavigator component has next and previous methods.
  - A RelationViewPlus component has next and previous methods.
- There is a data model associated with the objects that contain data. Each object has a set of values that can be indexed numerically.
  - A QueryNavigator component has symantec.itools.db.beans.jdbc.RecordDefinition objects. These objects obtain their data model on their own. The Table property is a property of RecordDefinition.
  - A RelationViewPlus component has symantec.itools.db.pro.Record objects. These objects obtain their data model from the SQL statements that define the RelationViewPlus.
- There is a relationship between objects in the view.
  - RecordDefinition components contained by a QueryNavigator component are related in that they meet the select criteria imposed by the Filter property of the QueryNavigator.
  - Records contained by a RelationViewPlus component are related in that they all meet the select criteria in the defining SQL statement.

For detail views, the objects (RecordDefinition or RelationViewPlus' Record) are further related in that they meet the join criteria.



## **Visual Cafe window**

Main development environment window that contains the menu bar and toolbars.

W

## Watch window

Debugger window that displays the value of a variable or expression you enter. The values update when you pause execution or step through code. You can also examine the contents of a class member. To watch a variable accessible to a method, drag a variable from the Variables window to the Watch window. You can modify values directly in this window and continue debugging without having to stop and restart the debug session.

To see the Watch window, choose View (or Window) ► Watch.



## **window**

Area that has no borders and no menu bar.

## **workspace**

Saved window arrangement. Your workspace is saved automatically while you work.

X

Y

z

## Environment Options (database tab)

The Database tab topic goes in the Visual Cafe help at the end of the file (but not in the obsolete section...). It's not part of the browse sequence in vcafe.hlp.

## IMPORTANT

Use these buttons in envopt.doc (not the ones currently in the topic):

```
{button Concepts,AL('Overview_of_Visual_Caf;Using_Visual_Caf_as_Your_Development_Environment',0,'')} {button See  
Also,AL('Environment_Options_Dialog_F1;Project_Options_dialog_F1',0,'')}
```

ALSO, in the footnotes # and A, take out “\_db”.



## Setting the database environment options

```
{button Concepts,AL('Database Development Edition_Extension_Overview',0,'')} {button See  
Also,AL('dbNAVIGATOR_pop_up_menu;dbNAVIGATOR_window',0,'')}
```

When you install the Database Development Edition of Visual Cafe, a Database tab is added to the Environment Options dialog box. You use this view to

- choose what API to use when Visual Cafe Database Development Edition generates code related to databases
- set the default data types for components in the dbAWARE Project and Table Wizards (Choose Database Columns page) and when you drag a column to a project from the dbNAVIGATOR window

### To access the Environment Options Database view

- Choose Tools ► Environment Options
- Database tab.

### To choose an API

- Select [JDBC API](#) or [dbANYWHERE API](#).

### To set a default data type for a component

1. Click the Component field of any data type to see the drop-down list of component data type options.
2. Choose an appropriate data type for your component.

### To return to the default data types

You can return to the defaults that shipped with Visual Cafe Database Development Edition.

- Click Set Defaults.

