



## PlanetJ Corporation

[\[support@planetJavaInc.com\]](mailto:support@planetJavaInc.com)

[\[info@planetJavaInc.com\]](mailto:info@planetJavaInc.com)

[\[www.PlanetJavaInc.com\]](http://www.PlanetJavaInc.com)

---

# WEB OBJECT WIZARD: Builders Guide [WOW 6.5 - 7.0]

## **License and Versions**

### **Getting Started**

[Starting WOW](#)

[Sign-on Fields](#)

[User Registration](#)

[Personal Info](#)

[Sign On Info](#)

[Understanding Development Accounts - Shared/Individual Accounts](#)

[Modifying the WOW Interface](#)

[Hiding the Side Steps](#)

### **Creating Connections**

[Creating a Connection Definition](#)

[Connection Spec](#)

[Options](#)

[Advanced](#)

[Specifying a Library List for a Connection](#)

[Connecting to Different Databases](#)

[Connecting to MySQL](#)

[Connecting to Oracle](#)

[Connecting to MS SQL Server](#)

[Connecting to Microsoft Access and Excel](#)

[Connecting to PostgreSQL](#)

[Connecting using jTDS jdbc driver \(open source\)](#)

[Working with Connections](#)

### **Creating Applications**

[Defining the Application](#)

[Basic](#)

[Display](#)

[Advanced](#)

[Alerts](#)

[Internal](#)

[MetaData Application Libraries](#)

[Creating Application Libraries](#)

[Using Application Libraries](#)

[Using the CPYWOWENV Command \(AS400/iSeries/IBM i Only\)](#)

[Making a Copy of MYSQL Metadata Library](#)

[Activating Alerts for an Application](#)

- [Configuring an Email Sever](#)
- [Configuring an Application to Send Email Alerts](#)
- [Controlling the Login](#)
- [Sign On Types](#)
- Creating Operations**
- [User Operations](#)
  - [Basic](#)
  - [Display](#)
  - [Advanced](#)
  - [Administration](#)
  - [Internal](#)
- [SQL Operations](#)
- [HTML Code Operations](#)
- [Other Operation Types](#)
  - [Operation Types](#)
- [Controlling the Display Order of Operations](#)
- [Setting a Next Operation](#)
- [Using a Please Wait Page](#)
- [Creating a Custom Please Wait Page](#)
- Selecting Records**
- [Basic SQL Queries Using the SELECT Statement](#)
  - [Microsoft SQL Server 2000 & Up](#)
  - [Microsoft Access](#)
- [Other Queries Using the SELECT Statement](#)
- [Setting Key Position Field to Select Unique Records](#)
- [SQL Tips](#)
  - [Case Sensitivity](#)
  - [Optional Values](#)
- Inserting Records**
- [Basic SQL Queries Using the INSERT Command](#)
- [Inserting Records without SQL Commands](#)
- [Inserting Records Using Parsing](#)
- [Joined Inserts](#)
  - [Restrictions](#)
  - [Transactions](#)
- Updating Records**
- [Basic SQL Queries Using the UPDATE Command](#)
  - [Using a WHERE clause with the UPDATE statement](#)
- [Joined Updates](#)
  - [Restrictions](#)
  - [Transactions](#)
- Deleting Records**
- [Basic SQL Queries Using the DELETE Command](#)
- [Deleting Rows Without SQL Commands](#)
- [Joined Deletes](#)
  - [Restrictions](#)
  - [Transactions](#)
- Field Descriptors**
- [Field Descriptor Manager](#)
  - [Library Functions](#)
  - [Table Functions](#)
- [Editing Rows Within the Field Descriptor Manager](#)
- [Basic Settings](#)

- [Display Settings](#)
- [Possible Value Settings](#)
- [Advanced Settings](#)
- [Authorization Settings](#)
- [Additional Settings](#)
- [Database Settings](#)
- [Field Descriptor Views](#)
  - [Table FD's](#)
  - [Shared FD's](#)
  - [Usage IDs](#)
  - [Search By](#)
  - [Quick Edit](#)
- [Prompting Using Field Descriptors](#)
- [WOW Features](#)**
  - [Derived Fields](#)
    - [Creating a Derived Field Descriptor](#)
  - [Parameters](#)
    - [SQL Prompt Parameters](#)
    - [Field Descriptor Prompt Parameters](#)
    - [Row Parameters](#)
    - [User Parameters](#)
    - [Usage ID Parameters](#)
    - [Table Parameters](#)
    - [Parameter Parameters](#)
    - [Context Parameter Parameters](#)
      - [Using Context Parameter parameters in Possible Values](#)
    - [Runtime Parameters](#)
    - [Request Parameters](#)
    - [Session Parameters](#)
    - [Special User Library Parameter](#)
    - [RowCollection Parameters \[Minimum Version: WOW 7.0\]](#)
    - [Defaulting Parameter Values](#)
  - [Operation Property Groups](#)
    - [AutoRun {}](#)
    - [Browser {}](#)
    - [Chart {}](#)
    - [Config {}](#)
    - [CSV {}](#)
    - [DetailDisplay {}](#)
    - [DisplayColumns {}](#)
    - [Email {}](#)
    - [FieldSet {}](#)
    - [Join {} \[PRO\]](#)
    - [LayoutDisplay {}](#)
    - [OperationLabels {}](#)
      - [Horizontal Parameters](#)
    - [OperationSettings {}](#)
    - [OptionalSignon {}](#)
    - [Paging {}](#)
    - [ParameterOperators {}](#)
    - [PDF {}](#)
    - [PleaseWait {}](#)
    - [PossibleValues {}](#)

- [ReportBreak {}](#)
- [SignOn {}](#)
- [SpooledFile {}](#)
- [SQLContext {}](#)
- [StoredProcedure {}](#)
- [Styles {}](#)
- [TableDisplay {}](#)
- [Tabs {}](#)
- [XLS {}](#)
- [Sorting](#)
  - [Controlling the Sorting Behavior](#)
  - [Changing the Column Heading](#)
  - [Changing the Header Style](#)
- [Associations](#)
  - [1-1 Association](#)
  - [1-Many Association](#)
  - [HTML Code Association](#)
    - [Full Field Rendering](#)
  - [HTML Reference Association](#)
  - [Associated Java Operation](#)
- [Creating Associations](#)
  - [SQL Association Example](#)
  - [HTML Code Association Example](#)
    - [Overview](#)
    - [Create Employee Operation](#)
    - [Create HTML Code Association Operation](#)
    - [Set the Association to a Field](#)
  - [HTML Reference Association Example](#)
- [Associated Inserts](#)
  - [SQL Associated Insert Example](#)
- [Associated Updates](#)
  - [SQL Associated Update Example](#)
- [Associated Deletes](#)
  - [SQL Associated Delete Example](#)
- [Join Associations](#)
- [Possible Values](#)
  - [Multiple Fields in Possible Values Drop Down](#)
  - [Possible Values and the – All – Value](#)
    - [Customizing the – All – Item](#)
    - [Further Customizing the – All – Item](#)
    - [Removing the – All – Item in a Search](#)
  - [Removing – Next – and – Previous – from Possible Value List](#)
- [PV Multiple Selects](#)
- [Possible Values Paging \(Next/Previous\)](#)
- [Possible Values Grouping \[Minimum Version: WOW 6.6 beta\]](#)
- [Possible Value Keys](#)
- [Possible Values Selector](#)
- [Possible Values Search](#)
  - [Steps to Utilize Possible Values Search Operation:](#)
  - [Using Possible Values Search to Populate Other Fields](#)
- [Auto Population of Fields](#)
- [Execution Groups](#)
  - [Create A Working Execution Group](#)



[Blob File Upload and Download](#)

[Set Up File Upload](#)

[Set Up File Download](#)

[Work Flow](#)

[Example 1](#)

[Example 2](#)

[Advanced Work Flow](#)

## **Context Menu**

[Controlling Actions in the Context Menu](#)

[Disabling the Context Menu](#)

[Removing Actions from the Context Menu](#)

[Showing Actions Only in the Context Menu](#)

[Suppressing Built-in Actions](#)

[Controlling the Context Menu Appearance](#)

[Using Different Action Descriptors](#)

[CSS Properties](#)

[Action Groups](#)

[Different Actions for Different Rows](#)

## **Auto Complete**

[Configuring Auto Complete Fields](#)

[Auto Complete Properties \[PRO\]](#)

[Auto Complete Advanced Configuration](#)

[Formatted Display Value](#)

[SQL-based Auto Complete \[PRO\]](#)

[Using SQL-based Auto Complete](#)

[Derived Fields](#)

[Auto Complete Fields in Rows](#)

## **Replacement Libraries**

[What is Replacement Library Support](#)

[For Example:](#)

[Four Ways to Implement Replacement Library Support](#)

[WOW Based](#)

[For Example:](#)

[Application Based](#)

[For Example:](#)

[User Based](#)

[For Example:](#)

[URL Based](#)

[For Example:](#)

[Replacement Library Implementation Precedence](#)

[For Example:](#)

[Please click here, WOW Builders Guide continues.](#)

# License and Versions

This guide includes information for all editions of WOW including WOW Community Edition, WOW Professional, and WOW Enterprise Edition. Your license agreement restricts you to using only features that you are licensed for. Features licensed to Professional or Enterprise are depicted as shown below:

WOW Edition	Identifier	Comments
WOW Professional Edition	[PRO]	This feature requires WOW Professional Edition.
WOW Enterprise Edition	[EE]	This feature requires WOW Enterprise Edition.

This guide may contain information for upcoming features not yet available to the user community. These upcoming features will be indicated with [WOW x.x] where xx identifies the version required.

From the WOW Builder screen, the key environment variables are shown below:

**Project Schema:** This is the MYSQL schema or the IBM i library that holds WOW metadata such as operations, connections, etc.

**Dev Schema:** Advanced usage only. Indicates a current target library or schema.

**WOW Version:** The current version of WOW code.

**License:** The license running this WOW instance. Values include COMMUNITY, PROFESSIONAL, or ENTERPRISE.



## Getting Started

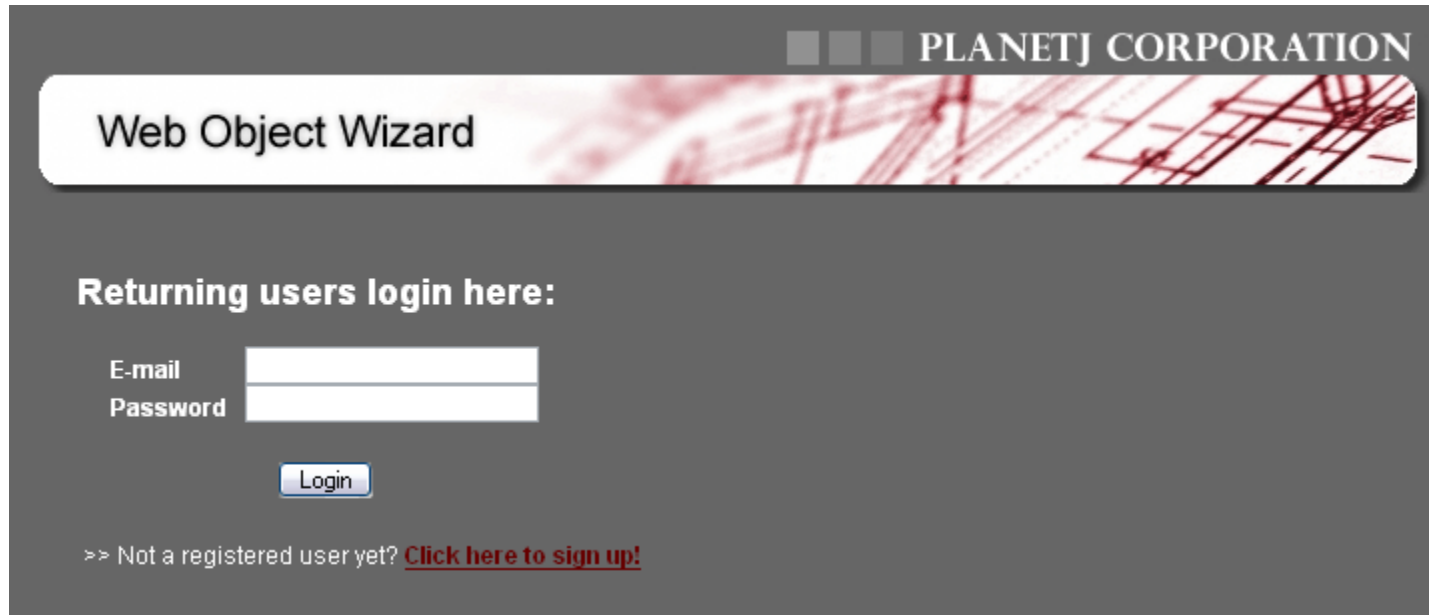
### Starting WOW

To start Web Object Wizard from the PlanetJ website, navigate to the following address:  
<http://www.planetjavainc.com/wow/WOWBuilder>

If you have WOW installed on a local computer or intranet, simply use your web browser point to the location where WOW is installed on your system. This would most likely be:

[http://my\\_server\\_name/wow65/WOWBuider](http://my_server_name/wow65/WOWBuider)

The *my\_server\_name* should be replaced with the name of your web server (e.g. localhost, 192.168.1.10, etc.). Version 6.5 is activated using the context *wow65*. Various versions of WOW can be accessed accordingly (for example, WOW 6.4 would use the context *wow64*). A sign-on page will appear once WOW has been started in the browser.



The screenshot shows the PlanetJ Corporation Web Object Wizard login interface. At the top right, the PlanetJ Corporation logo is displayed. Below it, a banner reads "Web Object Wizard". The main section is titled "Returning users login here:". It contains two input fields: "E-mail" and "Password". Below these fields is a "Login" button. At the bottom, there is a link: ">> Not a registered user yet? [Click here to sign up!](#)".

## Sign-on Fields

- **E-mail** (Required Field) - The e-mail address used during the registration process. The same information will be used to login to WOW.
- **Password** (Required Field) - The unique password specified during the registration process.

## User Registration

Clicking on the *Click here to sign up!* hyperlink will take you to the User Registration page. Throughout this guide as well as WOW, required fields will be indicated by a red asterisk (\*). After all relevant information is entered, click the *Sign Up* button to add the new registered information into the database. The specified e-mail address and password can now be used to log into WOW.

The screenshot shows a user registration form with a dark gray background. At the top right are 'Sign Up' and 'Cancel' buttons. The form is divided into three sections, each with a red header bar and a small icon:

- Personal Info:** Contains three text input fields: 'First Name\*' (with a red asterisk), 'Last Name\*' (with a red asterisk), and 'Work Phone #'. The first two fields are wide, while the third is narrower.
- Sign On Info:** Contains two text input fields: 'E-mail\*' (with a red asterisk) and 'Password\*' (with a red asterisk). Both fields are wide.
- Additional Settings:** Contains a 'Mode\*' label (with a red asterisk) and a dropdown menu currently showing 'Novice'.

At the bottom right of the form are another 'Sign Up' and 'Cancel' buttons.

### Personal Info

- **First Name** (Required Field) - The first name that will be used in conjunction with WOW.
- **Last Name** (Required Field) - The last name that will be used in conjunction with WOW.
- **Work Phone #** - The optional phone number used to contact the user.

### Sign On Info

- **E-Mail** (Required Field) - The e-mail address which will be used to log into WOW.
- **Password** (Required Field) - The password used to log into WOW. The maximum length is 10 characters. Passwords should be as unique as possible; it is recommended to use a combination of letters and numbers and is not a dictionary word.

## Understanding Development Accounts - Shared/Individual Accounts

A WOW development account (keyed by EMAIL) can be created as a "shared" account or an individual account. Accounts are completely separated from each other and do not share any resources such as connections, operations, etc. A "shared" account is a generic account where multiple developers share the same email such as AR\_REPORTS@MYCOMPANY.COM. An individual account is typically used by a single developer and all resources are separate (e.g. John@acme.com).

If an organization wishes to share WOW resources amongst multiple developers they should use a generic and shared account. Use individual accounts if development is to be restricted by individual with no need to share resources.

The account email and password can be changed in the WOW Utilities application using the "preferences" menu item. WOW Utilities can be access from the main WOW builder page under "Development Tools".

## Modifying the WOW Interface

### Hiding the Side Steps

To collapse (or hide) the side steps panel, click the *Hide Side Steps* menu option immediately below the header graphic.

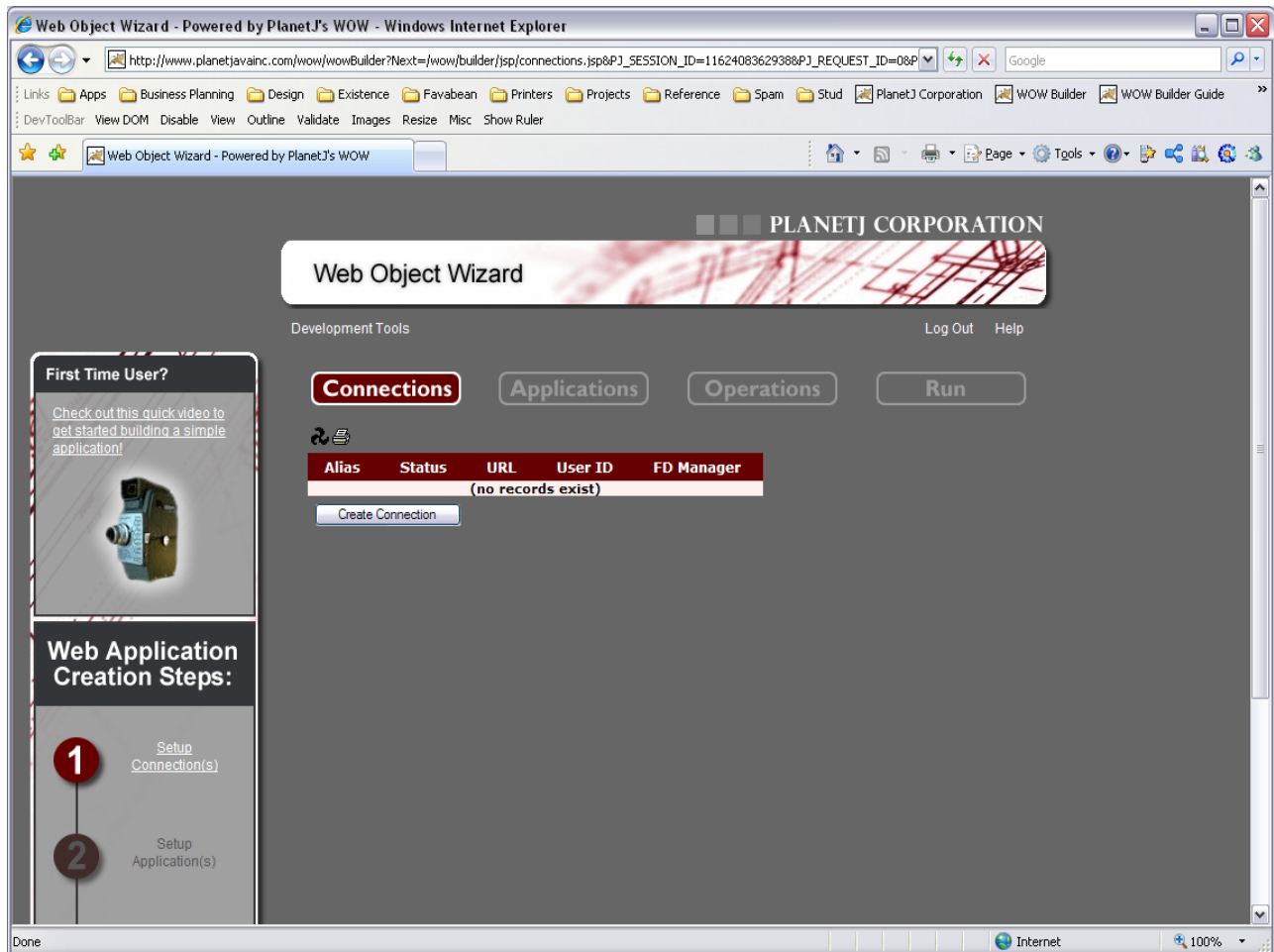


Once you click this option, the side steps panel will collapse and shift all of the screen contents to the left. Also, the menu option will change to say *Show Side Steps*. This option will reverse the changes and reveal the side steps panel.

# Creating Connections

## Creating a Connection Definition

After successfully signing onto WOW, you should see the main screen for Web Object Wizard:



Before creating an application, a database connection must be defined. This connection can be to any database that is compatible with WOW. To setup the new connection, click on either the *Setup Connections* link in the side steps panel or the *Connections* button in the toolbar. Then, click on the *Create Connection* button below the list of connections to bring up a screen similar to this:

**Connection Spec**

JDBC Driver\*

AS/400 Remote ▼

Alias\*

YOUR SYSTEM NAME

URL\*

jdbc:as400:YOUR IP ADDRESS

Properties

;trace=false;prompt=false;translate binary=true;sort=language

User ID\*
Password\*

**Options**

View Advanced Settings ☒

Auto Verify ☒

**Advanced**

Min. Connections

3

Max. Connections

10

Orphan Timeout (sec.)

1800

Clean Up Timeout (sec.)

9800

Connection Class

## Connection Spec

- **JDBC Driver** (Required Field) - The type of JDBC you will be using to connect to the database. This specifies the specific class that you will use to access the database.
  - AS/400 Native - Use when your data resides on the AS/400 and your application runs directly on the AS/400.
  - AS/400 Remote - Use when your data resides on the AS/400 and your application server does not run on the AS/400.
  - AS/400 Command & Program Call - For future support.
  - DB2 (Local) - Use when your data resides in DB2 and your application runs on the same server that contains DB2.
  - DB2 (Remote) - Use when your data resides in DB2 and your application runs on a different server.
  - MS Access/Excel (ODBC) - Use when your data resides in an Excel spreadsheet or a MS Access database connected through a System DSN.
  - MySQL - Use when your data resides in MySQL.
  - PostgreSQL - Use when your data resides in PostgreSQL.
  - ORACLE (Remote) - Use when your data resides in ORACLE.
  - SQL Server - Use when your data resides in Microsoft's SQL Server.

- **Alias** (Required Field) - Any text which uniquely identifies the database connection. Entries should be easily associated with the connection being created. The maximum entry is 50 characters. The Connection Alias is used when looking up field descriptors and normally should not be edited after its creation.
- **URL** (Required Field) - The URL of the JDBC database to be connected to. This URL will be specific to the type of database you are connecting to. This URL is where your database information is located.
- **Properties** - Specific properties that are set in the specific connection you have created. Refer to each JDBC driver for more information.
- **User ID** (Required Field) - An ID that will be used to connect to a specific database. This must be a valid user ID for the database that you will be connecting to. All database operations will be executed through this user ID.
- **Password** (Required Field) - The password which corresponds to the user ID used to connect to a specific database. This must be a valid password for the user ID you are using.

## Options

- **View Advanced Settings** - Shows the Advanced settings section.
- **Auto Verify** - Automatically verifies the connection settings by attempting to connect to the database upon insertion. This will return an error if it is unable to connect.

## Advanced

- **Min. Connections** (Required Field) - The number of connections that will be created when the application first starts up. The maximum value is 10. The default value is 2.
- **Max Connections** (Required Field) - The maximum number of simultaneous connections allowed for the database connection. The maximum number of connections used can have a significant effect on the performance of the system; the number will vary based on the power of the system. The default value is 10.
- **Orphan Timeout** (Required Field) - The maximum number of seconds that a database transaction is allowed to take. When a database transaction takes longer than the allocated time, it is terminated and the connection is made available for a new transaction. This prevents a database transaction from hanging and permanently tying up a connection.
- **Clean Up Timeout** (Required Field) - After the specified amount of seconds, the program will close and reopen its connection. Many databases only allow connections to remain open for a certain amount of time. This setting helps ensure that a connection will not time out, and if such occurs, it will be reopened.
- **Connection Class** - Used to enhance or modify the connection using a java class. For example, a java class could be written that changes the theme or header when running against test data rather than live data.

## How Connection Pools Work

Connection pooling has become the standard for middleware database drivers. The process of creating a connection, always an expensive, time-consuming operation, is multiplied in these environments where a large number of users are accessing the database in short, unconnected operations. Creating connections over and over in these environments is simply too expensive. The transaction profile for Web applications, probably the most common application in use today, is that of a large number of users performing short, discrete database operations. These applications usually perform work centered around creating a web page that will be sent back to the user's browser. Transactions are generally short-lived, and user sessions are often limited in time.



A connection pool operates by performing the work of creating connections ahead of time. In the case of a JDBC connection pool, a pool of **Connection** objects is created at the time the application server (or some other server) starts. These objects are then managed by a **pool manager** that disperses connections as they are requested by clients and returns them to the pool when it determines the client is finished with the **Connection** object. A great deal of housekeeping is involved in managing these connections. When a WOW Operation is executed, an unused connection from the pool is checked out, SQL is executed using the Connection, and then the Connection is returned to the available pool.

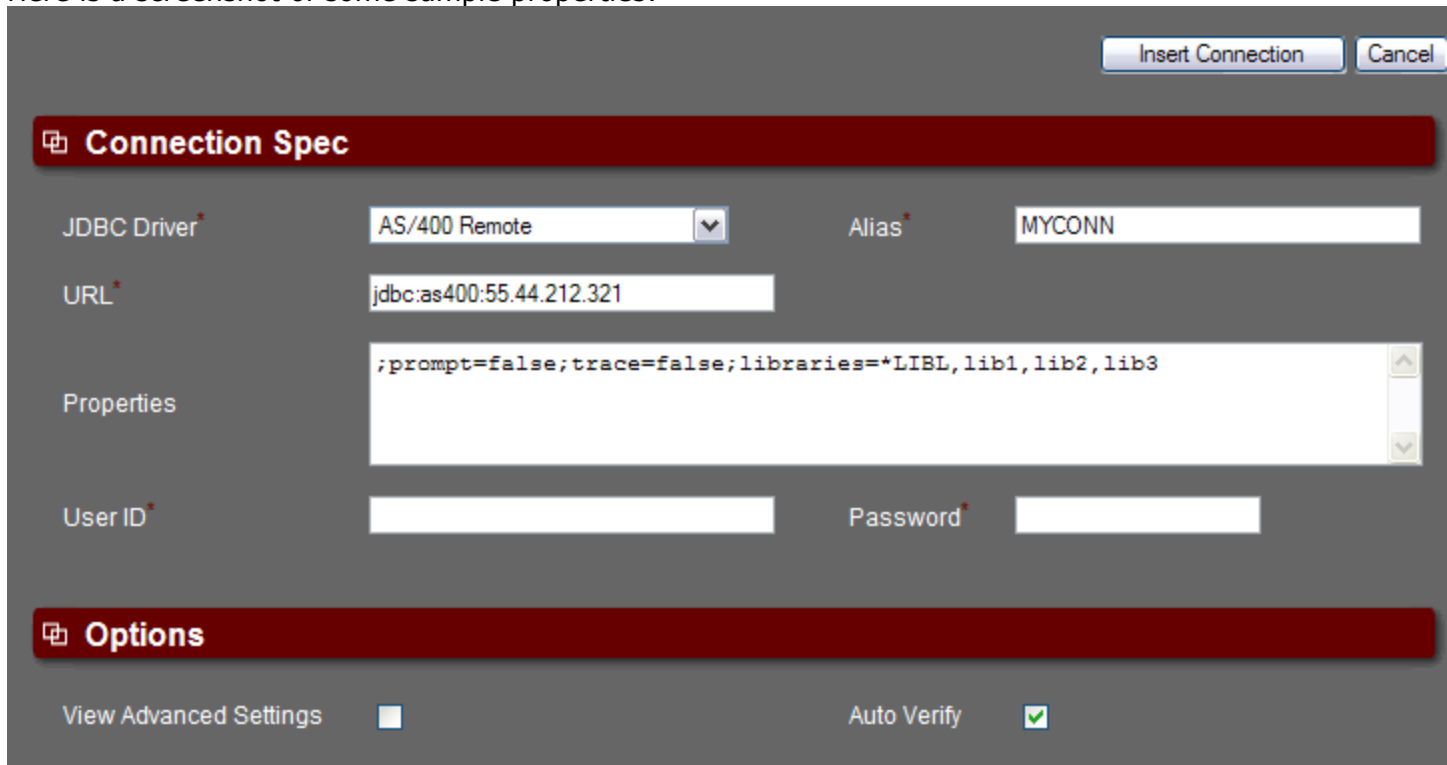
## Specifying a Library List for a Connection

For IBM i, one of the properties for a connection is the library list to be used. This can be especially useful when the connection is used for calling a stored procedure or SQL trigger. To specify the library list, append the "libraries" property to the connection's *Properties* field.

For example, the connection properties might be changed to:

```
;prompt=false;trace=false;libraries=*LIBL,lib1,lib2,lib3
```

Here is a screenshot of some sample properties:



The screenshot shows a 'Connection Spec' dialog box with the following fields and values:

- JDBC Driver:** AS/400 Remote
- Alias:** MYCONN
- URL:** jdbc:as400:55.44.212.321
- Properties:** ;prompt=false;trace=false;libraries=\*LIBL,lib1,lib2,lib3
- User ID:** (empty)
- Password:** (empty)

At the bottom, there are two checkboxes: 'View Advanced Settings' (unchecked) and 'Auto Verify' (checked).

With this example, the connection will append libraries *lib1*, *lib2*, and *lib3* to the end of the default library list.

**NOTE:** This example applies to the iSeries (AS/400). For other platforms, refer to the appropriate JDBC documentation.

IBM Toolbox for Java Details: <http://publib.boulder.ibm.com/infocenter/iseries/v7r1m0/index.jsp?topic=%2Frzahh%2Fpage1.htm>

## Connecting to Different Databases

WOW can connect to any database that supports a JDBC 2.0 driver. This allows WOW applications to seamlessly combine data from any corporate data repository regardless of its location and RDBMS vendor. Below is a list of specific databases and what URL's are needed to connect to each of them. Replace the IP address listed with the IP address used to connect to your own database.

- **AS/400 (iSeries)** - jdbc:as400:66.166.144.20
- **SQL Server** - jdbc:microsoft:sqlserver://66.166.144.20
- **Oracle** - jdbc:oracle:thin:@66.166.144.20:1521:METADATA
- **DB2** - jdbc:db2://66.166.144.20/DB\_NAME
- **MySQL** - jdbc:mysql://localhost/pjsys64
- **ODBC** - jdbc:odbc:Data Source Name
- **PostgreSQL** - jdbc:postgresql://66.166.144.20/DB\_NAME
- **JTDS** (used for SQL Server) - jdbc:jtds:sqlserver://66.166.144.20/DB\_NAME

**NOTE:** Replace *METADATA* (Oracle), *DB\_NAME* (DB2, PostgreSQL), or *pjsys64* (MySQL) with the your database name. For ODBC, replace *Data Source Name* with the name of your DSN (which must be a system DSN that points to the desired Access database on your machine).

### Connecting to MySQL

The connection properties that are unique to MySQL are the *URL*, *Driver*, *User ID*, and *Password*. They must be exactly as follows:

- URL - **jdbc:mysql://localhost/pjsys64** (where *localhost* is your IP address and *pjsys64* is your database)
- JDBC Driver - **MySQL** (com.mysql.jdbc.Driver)
- User ID - **WOW**
- Password - **wow**

Here is a sample MySQL connection:

Insert Connection Cancel

### Connection Spec

JDBC Driver\* MYSql Alias\* LOCAL\_MYSQL

URL\* jdbc:mysql://localhost/pjsys64

Properties

User ID\* WOW Password\* ●●●

### Options

View Advanced Settings ☐ Auto Verify ☒

If you want different properties for *User ID* and *Password*, you will have to add records in the *mysql.user* table. Please consult the [MySQL Reference Manual](#) for details on how to create new user accounts.

MySQL Manuals: <http://dev.mysql.com/doc/>

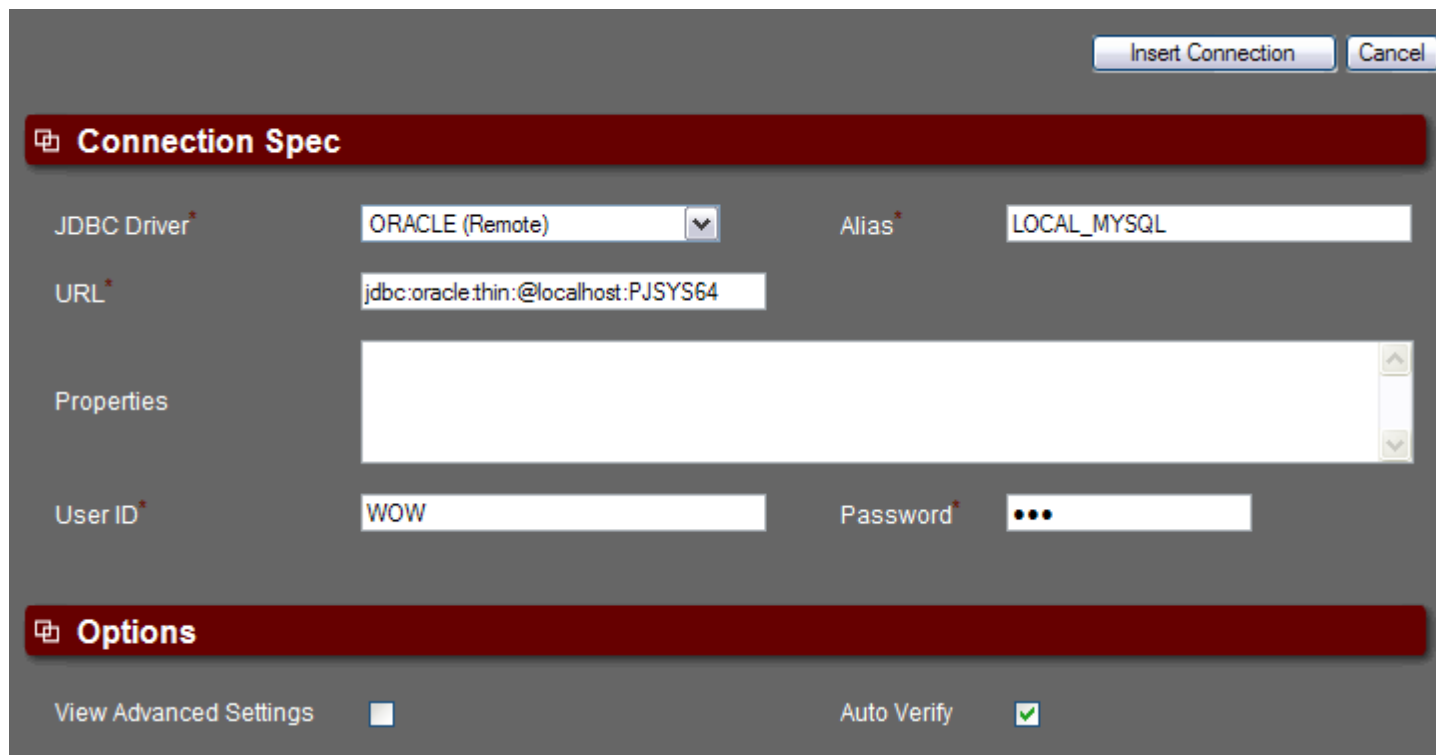
**NOTE:** *User ID* and *Password* are case sensitive in MySQL.

## Connecting to Oracle

The connection properties that are unique to ORACLE are the *URL*, *Driver*, *User ID*, and *Password*. They must be exactly as follows:

- URL - **jdbc:oracle:thin:@localhost:PJSYS64** (where *localhost* is your IP address and *PJSYS64* is your database)
- JDBC Driver - **ORACLE (Remote)** (oracle.jdbc.driver.OracleDriver)
- User ID - **Any valid user ID**
- Password - **Any valid password**

Here is a sample Oracle connection:



Insert Connection Cancel

### Connection Spec

JDBC Driver\* ORACLE (Remote) Alias\* LOCAL\_MYSQL

URL\* jdbc:oracle:thin:@localhost:PJSYS64

Properties

User ID\* WOW Password\* ●●●

### Options

View Advanced Settings ☐ Auto Verify ☒

Oracle SQL Docs: [http://docs.oracle.com/cd/B19306\\_01/server.102/b14200/toc.htm](http://docs.oracle.com/cd/B19306_01/server.102/b14200/toc.htm)

**NOTE:** Since Oracle has a different jar for each of their releases, Oracle users may have to manually upgrade their JDBC jar with the version for their database. WOW's default `ojdbc14.jar` needs to be the same one as the one included in your installed Oracle version (e.g. `oracle/product/10.2.0/db_1/jdbc/lib/ojdbc14.jar`).

## Connecting to MS SQL Server

WOW supports MS SQL Server 2000 and above, which is supported by the latest JDBC driver. You may have connection problems if trying to connect to a version under MS SQL Server 2000.

URL - **jdbc:microsoft:sqlserver://hostname:port;databasename=dbname**

Example: `jdbc:microsoft:sqlserver://192.169.1.71:1433;databasename=Northwind`

Web Object Wizard must be able to resolve the host name or specified IP. Port 1433 is the default port and is optional. A particular database instance can be identified by *databasename*.

**Connection Properties** can be supplied on the connection to influence driver behavior. <http://msdn.microsoft.com/en-us/library/ms378988.aspx>

For example, the following may be specified on the connection properties: **`responseBuffering=adaptive;sendStringParametersAsUnicode=false;`**

**SQL Server Docs:** <http://msdn.microsoft.com/en-us/library/bb545450.aspx>

## Connecting to Microsoft Access and Excel

Web Object Wizard supports Microsoft Access and Excel 2000 or above, which is supported by the latest JDBC-ODBC driver. You may have connection problems if trying to connect to a version under Microsoft Access/Excel 2000. To connect to Access or Excel, you must create an ODBC System Data Source **on the same system** that you have your application server installed. The DSN must be pointed to your desired Access database or Excel worksheet.

URL - **jdbc:odbc:data\_source\_name**

Example: jdbc:odbc:Northwind (connecting to a MS Access database Northwind)

WOW must be able to resolve the data source name.

See the Interfacing WOW with Excel document for more details.

## Connecting to PostgreSQL

The connection properties that are unique to PostgreSQL are the *URL*, *Driver*, *User ID*, and *Password*. They must be exactly as follows:

- URL - **jdbc:postgresql://hostname:port/databasename** (where hostname is your IP address, port is an optional port # (default is 5432), and *databasename* is the database)
- JDBC Driver - **PostgreSQL** (org.postgresql.Driver)
- User ID - **Any valid user ID**
- Password - **Password for above User ID**

Here is a sample PostgreSQL connection:

The screenshot shows a 'Connection Spec' dialog box with a dark red header. It contains the following fields:

- JDBC Driver:** A dropdown menu with 'PostgreSQL' selected.
- Alias:** A text field containing 'MY\_POSTGRES\_CONN'.
- URL:** A text field containing 'jdbc:postgresql://66.166.144.20:5433/My'.
- Properties:** A large, empty text area.
- User ID:** A text field containing 'pjuser'.
- Password:** A text field with masked characters (dots).

Optional Properties include: `ssl`, `sslfactoryarg`, `compatible`, `protocolVersion`, `loglevel`, `charSet`, `allowEncodingChanges` and `prepareThreshold`.

**NOTE:** You'll need to download the latest postgresql jar from <http://jdbc.postgresql.org>

## Connecting using jTDS jdbc driver (open source)

The open source jTDS driver can be used to connect to SQL Server.

The connection properties that are unique to jTDS are the *URL*, *Driver*, *User ID*, and *Password*. They must be exactly as follows:

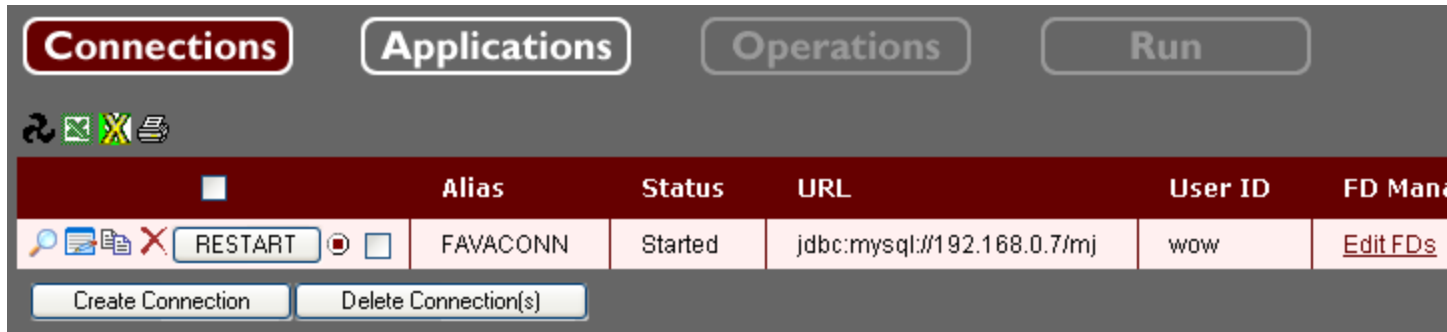
- URL - **jdbc:jtds:sqlserver ://<server> [:<port>][/<database>]** (where *server* is your IP address, *port* is an optional port # (specify if not using the default), and *database* is the database)
- JDBC Driver - **jTDS SQL Server JDBC Driver** (net.sourceforge.jtds.jdbc.Driver)
- User ID - **Any valid user ID**
- Password - **Password for above User ID**

**NOTE:** For connection properties and more details, see <http://jtds.sourceforge.net/faq.html>

**NOTE:** You'll need to download the latest jTDS jar from <http://jtds.sourceforge.net>

## Working with Connections

You can manipulate your database connections, using the direct action buttons as well as other actions on each connection row. Below is a screen shot of the *Connections* screen with brief descriptions of available actions:



- **View Connection** - Allows a user to view a selected connection that has been previously created.
- **Edit Connection** - Allow changes to be made to the selected database connection.
- **Copy Connection** - Allows a user to copy the selected database connection.
- **Delete Connection** - Deletes the selected connection. If you delete a connection, you must make sure any operations that reference that connection are updated to reference a working connection. Deleted connections cannot be restored.
- **Stop Connection** - Closes down the selected connection. All cached field descriptors and rows related to this connection will be cleared. This link should be used with caution; any applications running that use this connection (including the *Field Descriptor Manager*) will no longer function once the connection is stopped.
- **Start Connection** - Starts the selected connection if it is not already started.
- **Restart Connection** - Shuts down the connection (if started) and then restarts it. All cached field descriptors and rows related to this connection will be cleared.
- **Create Connection** - Creates a connection.
- **Delete Connection(s)** - Deletes selected connection(s).
- **Edit FDs** - Opens a new browser window containing the Field Descriptor Manager application for the selected application.



# Creating Applications

## Defining the Application

After creating a connection, follow the *Setup Application(s)* hyperlink in the side panel or the *Applications* button in the toolbar. You will then see the *Application Creation* screen.

The screenshot shows the 'Application Creation' screen with the following sections:

- Basic**: Fields for Name, Description, Connection (dropdown), Initial Operation (dropdown), Sign On Type (dropdown), and Sign On Operation (dropdown). Buttons: Insert Application, Cancel.
- Display**: A text area for the layout template containing the code: `LayoutDisplay(header::top::body::template::footer::pages::)`. Fields for Theme (dropdown) and Company Name. Buttons: Insert Application, Cancel.
- Advanced**: Fields for Subclass, JSP File (radio buttons), Optional Sign On (checkbox), and Auto Run Status (dropdown). Buttons: Insert Application, Cancel.
- Application Alert**: Field for Enable Application Alert (checkbox). Buttons: Insert Application, Cancel.
- Internal**: Field for ID (text box). Buttons: Insert Application, Cancel.

### Basic

- **Name** (Required Field) - The title or name for the application.
- **Description** - A brief description of the application.

- **Connection** (Required Field) - An alias for the system on which the application is located. This must be one of the connections already created. All connections, unless otherwise specified, will use this connection.
- **Sign On Type** (Required Field) - The type of security that your application will require.
- **Initial Operation** - The initial operation that the application runs when a user first executes this application.
- **Sign On Operation** - The user created Authentication Operation, if any, the application will run when a user signs on to this application.

## Display

- **Properties** - Parameters to customize the look and some of the features of WOW. Specifying different Property Groups allows for custom JSP's and many other possibilities. The default Property Group, Layout Display, allows for a custom header, footer, body, and template. Only users familiar with JSP programming should attempt to use these fields.
- **Theme** (Required Field) - This will change the look and feel of the WOW application. This includes backgrounds, links, buttons and general appearance of the application. WOW comes pre-installed with several themes.
- **Company Name** - The text entered in this field will be displayed in the left hand side of the WOW header.

## Advanced

- **Subclass** -[PRO] The class name of the application's main servlet. Unless the application uses custom programming, this field should be left blank and the default WOW servlet will be used. To use a subclass named *MySubClass* in the com.mypkg package, specify: `com.mypkg.MySubClass`.
- **JSP File** - [PRO]The name of the application's main JSP. Unless the application uses custom programming, this field should be left blank and the default WOW JSP will be used. To use a JSP named *myJsp* in the "...\\webapps\\wow65\\user\\planetj\\jsp" folder, specify the following: `/user/planetj/jsp/myJsp.jsp`.
- **Optional Sign-On** (Required Field) - For future support. If this option is selected, an optional sign-on box will appear in the upper left hand side of the TOC.
- **Auto Run Status** - Enabling Auto Run allows you to automate the distribution of data via email (for Auto-run Email operations) and to automatically log incoming email.

## Alerts

- [EE] See section **Activating Alerts for an Application** below for details. NOTE: [Minimum Version: WOW 7.0]

## Internal

- **ID** (Required Field) - The ID of the application. This ID can be used for a reference to the specific application. This field is set automatically by WOW.

After filling in the values for each field, click the *Insert Application* button to create the application. This will bring you back to the main screen where you can begin creating operations for your customized WOW application.

## MetaData Application Libraries

By default, the metadata for all applications created using WOW is stored in a single library (PJUSER64) or (PJUSERxx) depending on your version of WOW. For more complex environments with multiple applications with varying delivery schedules you can implement multiple . For example, you may have two WOW applications running on your development box and you wish to move one of these applications to a production environment. Since metadata records for both applications are stored in the same files, you will, therefore, have to copy records only corresponding to the application you want to move over to your production environment without copying the records for the application that you do not wish to move.

On the other hand, if your application were stored in two different libraries on your development environment, then copying a single application to production would be easy – you could just copy all of the files in the application's library over to production. In general, each related group of WOW applications can be stored in its own application library to facilitate any future moves or migrations. However, if you have no need to move applications independently of each other, they can all exist in the same library. The recommendation is to use a single metadata library unless you experience staging problems.

## Creating Application Libraries

The default application library (PJUSER64) was installed on the WOW metadata system as part of the WOW installation process. In order to use any other application library, that library must exist on the WOW metadata system. There are two ways of creating a new application library. You could repeat the steps from the WOW general installation which installed the PJUSER64 library, except this time give the installed library a different name. An alternative is to create a new copy of the PJUSER64 library. Copying an existing application library will of course copy all applications in that library. Copied applications are independent of the original applications and can be modified or deleted without affecting the original applications.

## Using Application Libraries

When starting the WOW builder, the application library to be used can be specified directly in the URL. For example, the following link would start the WOW builder using MYAPP as the application library:

```
http://www.planetjavainc.com/WOWBuilder?_pj_lib=MYAPP
```

If no library is specified, the default application library used is PJUSER64. To start the WOW builder in general using a particular application library, you append the WOW builder URL with the string:

```
?_pj_lib=<APPLIB>
```

The ? designates the start of the URL parameters, *\_pj\_lib* is the name of the parameter and *<APPLIB>* is the name of the application library in which all WOW metadata should be stored. If one or more parameters are already present in the URL, replace the ? with the parameter separator &:

```
&_pj_lib=<APPLIB>
```

Each application library contains its own applications, connections, operations, field

descriptors, and user logins.

**NOTE:** Any user logins, connections, applications, operations, and field descriptors created with the WOW builder in one application library cannot be accessed from any other application library.

### Using the CPYWOWENV Command (AS400/iSeries/IBM i Only)

If the metadata system that you are using is AS400/iSeries, you can use the CPYWOWENV command provided in the PJSYS64 folder. This command copies a specified WOW library to a new library and gives you the option to clear the files.

From the CL Command Prompt, enter **PJSYS64/CPYWOWENV** and press F4. You should see the following screen:

```
                                COPY WOW ENVIRONMENT (CPYWOWENV)

Type choices, press Enter.

SOURCE LIBRARY . . . . .  Character value
DESTINATION LIBRARY . . . . .  Character value
CLEAR DATA (Y OR N) . . . . .  Character value
```

Enter the source library (in most cases, this would be the PJUSER64 library). Enter the name of the destination library. Specify whether or not the file data should be cleared or not and press *Enter*. You have now successfully created an Application Library.

### Making a Copy of MYSQL Metadata Library

If your WOW metadata is stored in MYSQL, you can use the MYSQL Administrator or Workbench Application to save and restore a copy. You simply need to save a copy of your source schema/library such as PJUSERxx and restore it with a name such as MyProductionApp. Use this link for detailed directions on how to save and restore MYSQL objects. [MYSQL Backup and Restore](#)

## Activating Alerts for an Application

NOTE: [EE][Minimum Version: WOW 7.0]

An application can be configured to send email alerts when failures occur within WOW for the specific application. The alert email provides information identifying the failing application, the application user, the failing operation, an approximate URL used, browser information, error details, etc. The alert feature will try to limit alerts so that only one alert is sent per day for a particular error.

Before any application can be configured to send alerts, you will need to configure at least one email server for WOW to use to send email alerts:

### Configuring an Email Sever

[EE] To add and email server entry for use by application alerts:

- Bring up the WOW Utilities menu from the WOW Builder (Development Tools > WOW Utilities).
- From WOW Utilities, bring up the email server entries (Other > Email Server Entries - Outgoing Email).
- If no email server entry is defined, continue with the remaining steps to add an email server entry.
- Click on Add Entry.

**Other Control Settings**

Control ID 
Default Server Entry?

**SMTP Mail Server Settings**

Server IP Address 
SMTP Port Number

**Security and Authentication**

Connection Security 
Authentication Required ☒

User ID

Password

**E-mail Address Settings**

To Address


From Address

- Set the configuration settings to match your email server, then click on Add Entry:
  - **Other Control Settings:**
    - Control ID (internal ID for the server entry)
    - Default Server Entry? - Only 1 entry can be designated as the default configuration to use.
  - **SMTP Mail Server Settings:**
    - Server IP Address - Set to the IP address of your email server.
    - SMTP Port Number - Set the port number to use. The default is 25 for outgoing email.
  - **Security and Authentication:**
    - Connection Security - Set the security to match your email server (Never, SSL, TLS). The default is Never.
    - Authentication Required - Does your email server require authentication (user ID, password) for outgoing email?
    - User ID - Set the email server user ID if authentication is required.
    - Password - Set the email server password if authentication is required.
  - **Email Address Settings (used for testing the configuration):**
    - To Address - The to ID to use for testing the email connection.
    - From Address - The from ID to use for testing the email connection.

## Configuring an Application to Send Email Alerts

[EE] Once we have a use-able email server entry in place, any application can be configured to send an alert message when failures occur.

- From the WOW Builder, bring up the list of applications.
- Edit the appropriate application entry:
- Move down to the Application Alert section and click on "Enable Application Alert" to expand that section:



The screenshot shows a web interface for configuring application alerts. It has a dark red header with the title 'Application Alert' and a small icon. Below the header, there are several fields: 'Enable Application Alert' with a checked checkbox, 'Email Server' with a dropdown menu showing 'smtphost.qualcomm.com(sonar)', 'Email From ID' with a text box containing 'alert@mycompany.com', 'Email To ID' with a text box containing 'support@mycompany.com', and 'Email Alert Title' with an empty text box.

- Fill in the appropriate configuration settings and click on Update Application:
  - Enable Application Alert - enables alerts and expands this section to show the other fields.
  - Email Server - Set to the email server entry to use for sending emails. If not set, the default entry (if defined) is used.
  - Email From ID - the From ID to use for sending alerts
  - Email To ID - One or more comma separate email ID's to receive the alerts
  - Email Alert Title - optional field to change the email title from it's default.

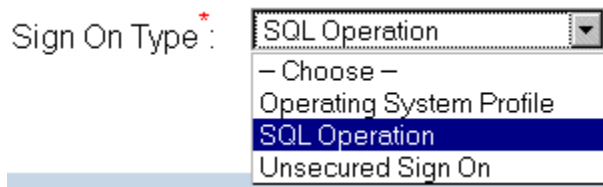
NOTE: Generally, validation type errors, including SQL errors due to improper operation configuration, do not trigger alerts.

## Controlling the Login

WOW allows users to login using different security schemes. This is important for applications that require different forms of user authentication.

### Sign On Types

- **HTTP Referrer**
- **Local Users Only**
- **Local Users Only or Operation System Profile**
- **Operating System Profile** - Users are required to sign on with a user ID and password before using the application. The user ID and password must be recognized by the database or operating system. The actual database access does not use this user ID, it uses the one specified in the connection definition.
- **Personal Connection Pool** - The *Personal Connection Pool* sign on validates a user ID and password against the database, much like the *Operating System Profile* sign on. However, when an application uses the *Personal Connection Pool* sign on method, all database accesses by that application will be tied to the profile of whichever user has signed onto the application and requested that database access. All other sign on methods use a shared pool of database connections when accessing the database – this can significantly improve performance but means that the database cannot determine which particular user is accessing it, only which application is doing the access. This sign on type should be selected when the database needs to know which user is accessing it.
- **SQL Operation** - Users must provide authentication information based upon the fields specified in an SQL operation. A logical choice for these fields would be the user ID and password; however, this option allows increased flexibility in that you can choose any field in a file to authenticate against. For example, you may use a single PIN field instead of the standard user ID and password combination.
- **Unsecured Sign On** - Users are not required to sign on to the application – anyone who knows the application's URL can use it. This is the default selection.
- **User List Sign On**





# Creating Operations

## User Operations

Each application created with WOW contains different operations. These operations are the backbone of any application created with WOW. There are many different types of operations that can be created with WOW. Examples of these types are: SQL, HTML Code, and Execution Group. Each available operation type will be described in greater detail in this chapter.

To add operations to an application, select the application name from the list of applications on the main menu. Next, follow either the *Setup Operation(s)* hyperlink in the side panel or the *Operations* button in the toolbar. This will display all of the operations in the application. Choose the *Create Operation* button to create a new operation.

The operation creation screen allows you to specify several different attributes of the operation.

### Basic

- **Label** - A unique name identifying the application for the user that appears in a list with all of the operations that have been created for the application.
- **Title** - The title that will be displayed when the user is viewing the list of operations.
- **Operation Type** - The type of operation you would like to use.
- **Description** - A brief description of the purpose of the operation.
- **Operation Code** - The actual code that will run when the user selects this operation. For SQL Operations, this must be a valid SQL statement for the database the application is connecting to. Incorrect code may cause WOW to return unfavorable results.
- **Instructions** - Text that will be shown to the user when the Operation is executed. This text is shown if the operation is an SQL statement that contains "where <column\_name> = ?" where <column\_name> is a valid column name from the indicated table. It may include details on what the operation does, how it is run, etc. This field may include actual HTML code to enhance the formatting. For example, you may want to make the instructions stand out by specifying the text be heading 1: "<h1>Instructions for the Operation</h1>".
- **Database Explorer button** - Launches an application in a separate window to assist the user with building an SQL statement (for the Operation Code field). It provides a list of fields for a specified table and provides options to help build a basic SQL statement. NOTE: [Minimum Version: WOW 7.0]

Basic

Label

Customers

?

Title

Sample Customers

Operation Type

SQL

?

Description

View a sample database

Operation Code

SELECT \* FROM QIWS.QCUSTCDT

Database




Instructions

?

Output Connection Alias

?

## Display

- **Allow Details** - Determines whether or not the *Details* button  is shown in the results table. This button allows the user to view the contents of one row in detail. By default, the *Details* button is shown.
- **Allow Inserts** - Determines whether or not the *Insert* button is shown in the results table. This button allows users to insert new rows into the table. By default, the *Insert* button is shown.
- **Allow Updates** - Determines whether or not the *Update* button  is shown in the results table. This button allows users to update the contents of a row. By default, the *Update* button is shown.
- **Allow Deletes** - Determines whether or not the *Delete* button  is shown in the results table. This button allows users to delete a row from the database. By default, the *Delete* button is not shown.
- **Display Group** - Determines how WOW separates different operations. All operations with the same *Display Group* will be grouped together (in the table of contents or drop down navigation area) when the application is run. Specifying a *Display Group* allows related operations to be displayed next to each other. Any name that would help with grouping the operations will work in this field.
- **Display Order** - The order in which the operations should be displayed. Operations with lower display orders are displayed before operations with larger display orders. *Display Order* also determines the order of the *Display Groups*. Within a given

*Display Group*, the lowest number in that set is shown first. The lowest number from each *Display Group* is the used to determine the order of the *Display Groups*.

- **Display Columns** - Shows which columns should be displayed. Typically, an SQL operation will return multiple columns from a database table (or tables), and will display all of these columns to the user. If you only want to display some of the returned columns, list those columns in this field (separated by commas) and only those columns will be shown.
- **Properties** - Controls specialty features of the operation such as page breaks, columns displayed, button text, business graphs, etc.

Display

Allow Details

☒

Allow Inserts

☒

Allow Updates

☒

Allow Deletes

☐

Display Columns

Display Rule

-- None --

Display Location

☒

-- None --

Display Group

☒

-- None --

Display Order

Properties

DisplayColumns{ results;; details;; }

DetailDisplay{

addButtonsURI;;

cancel;;

copyURI;;

editButtonsURI;;

insert;;

insertText;;

maxInputWidth;;

nextText;;

button locations;;

colCnt;;

delete;;

editURI;;

insertAndCopy;;

insertURI;;

maxInputWidthSum;;

previousText;;

buttonJustify;;

colonAfterLabel;;

deleteText;;

grid;;

insertAndNew;;

label justify;;

nextAndPrevious;;

printURI;;

## Advanced

- **Connection Alias** - The database connection that was set up for the operation. By default, operations use the connection alias specified in their application. If an operation needs to use a different connection, select it in this field.
- **Operation Class** - [PRO] A custom java class allowing you to override the execute method to change the default operation behavior. For example, if a table is displayed by the operation and there are no rows to display, you could instead display the add screen.
- **Row Count** - Specifies how many rows are displayed in the results table. If the number of results is greater than this value, links are generated on the results table allowing the user to page through it. The default is set at 50 rows. This field should be adjusted based on your system performance and connection speed.
- **Row Coll. Class** - [PRO] Specifies which *RowCollection* subclass this particular

operation will use. To use a row collection named *MyRowColl* in the *com.mypkg* package, specify the following: *com.mypkg. MyRowColl*

- **Row Class** - [PRO] Specifies which *Row* subclass this particular operation will use. To use a row subclass named *MyRow* in the *com.mypkg* package, specify the following: *com.mypkg. MyRow*
- **Parameters JSP** - [PRO] Parameters refers to the search prompts given when using dynamic prompting in an SQL statement. Use this field to specify a custom parameters JSP to replace the default for that particular operation. To use a JSP named *myJsp* in the "...\\webapps\\WOW65\\user\\planetj\\jsp" folder, specify the following: */user/planetj/jsp/myJsp.jsp*
- **Caching Level** - Sets the caching level of the operation. Caching deals with how WOW stores the information so it can be used later. It is similar to any other caching that you would use with a web browser, etc.
  - Cache for 1 Day - Checks and stores cache for 1 day.
  - Cache for 1 Hour - Checks and stores cache for 1 hour.
  - Cache for 1 Week - Checks and stores cache for 1 week.
  - Cache for 15 Minutes - Same as check cache and store results, except the results are only stored for 15 minutes. After that they are removed from the cache.
  - Cache for 30 Minutes - Same as check cache and store results, except the results are only stored for 30 minutes. After that they are removed from the cache.
  - Check results after a DB read - After results are read from the database store them in the cache.
  - Check cache and cache results - Check the cache before reading the database; if the database is read, then store the results in the cache afterwards.
  - Check cache on DB read - Check the cache before reading the database.
  - No caching - Do not use caching for this operation.
- **JSP File** - [PRO] The JSP file to use for displaying the results of this SQL operation. This field currently gives five choices:
  - Existing List or Insert - If any records are returned, then list them. If not, go to insert view.
  - Header/Detail Reports - Return results in a report format with header, details, and footer.
  - Single Row Edit - Returns a single record in the details view with editable fields.
  - Single Row View - Returns a single record in the details view.
- **Details JSP** - [PRO] Determines which JSP file to use when displaying the details of a single result of this operation. To use a JSP named *myJsp* in the "...\\webapps\\WOW64\\user\\planetj\\jsp" folder, specify the following: */user/planetj/jsp/myJsp.jsp*
- **Parent Operation** - Used to identify the tab parent operation when defining tab operations.
- **Depends On** - *Advanced use only.*
- **Usage ID** - Can be assigned to identify a particular usage. This can be used to dynamically copy data with the same usage ID. UI code can be written to look for fields with a particular usage ID such as an electronic store where JSPs might anticipate a *RowCollection* coming in with usage fields for ID number, image, price, etc.
- **Execution Rule** - Normally, when a user chooses to run an operation containing user parameters, the operation is not run until after the user fills in values.
  - Prompt then Execute - When a user chooses to run an operation containing user parameters, the operation is not run until after the user fills in values. This is the default behavior.

- Execute then Prompt - When a user chooses to run an operation containing user parameters, the operation is run first (using default values for the parameters) then the user is prompted.
- Execute Only - When a user chooses to run an operation containing user parameters, the operation is run without ever prompting the user. This type of operation works well for associated inserts, updates, and deletes where the prompt parameters are actually set within code or grabbed from some other source (like default values) where user prompting is not needed.
- **Next Operation** - Allows the selection of another operation to execute when the current operation completes. The current operation completes when a row is inserted, updated, or deleted. At that time, the next operation is executed.

## Administration

- **Security Type** - [PRO] The type of security measures to use.
- **Security Level** - [PRO] Security level is used in conjunction with the *Security Type* feature.
- **Auto Run Op.** - Allows you to specify an operation that will automatically run on a set schedule. The pull-down for this field displays any available Auto Run operations created within the application.
- **Execute Authority Operation** - [PRO] Used to limit which users can view and run the operation. All Authorization Operations defined for the current application should appear in the drop down selection. If no operation is selected, all users will be authorized to execute this operation.
- **Auto Run Status** - This field only applies to Auto Run operations. *Disabled* means do not run this operation, even when Auto Run is enabled for the application. *Enabled* means always run this operation when Auto Run is enabled for the application. *Production Only* means run this operation in the production environment only. *Development Only* means run this operation in the development environment only.

## Internal

- **Operation ID** - The ID used for categorizing and tracking operations. WOW assigns this value internally and cannot be changed.
- **Application** - The application of which the operation is a part or member. Use the drop down to change the application to which this operation belongs.

## SQL Operations

SQL operations are one of the more important operation types that can be created with WOW. SQL stands for Structured Query Language. SQL is used to manipulate the data in a database. Although SQL is not a difficult programming language to understand, it is very extensive. The only SQL covered in this guide will be examples on how SQL works with WOW. If you are unfamiliar with SQL, you will still be able to follow the examples in this manual; however, it is highly recommended that you review the following SQL manuals and tutorials before using WOW.

- **W3 Schools SQL** (<http://www.w3schools.com/sql/default.asp>)
- **SQL Course** (<http://www.sqlcourse.com/>)
- **PDF SQL Tutorial** ([http://www.thinkbrown.com/programming/sql\\_tutorial.pdf](http://www.thinkbrown.com/programming/sql_tutorial.pdf))

**NOTE:** The PDF SQL Tutorial link opens a Portable Document Format (PDF) file and you must have Adobe® Reader® or Adobe Acrobat® to view this file. To download a copy of Adobe Reader from the Adobe website, click on the icon below.



**SQL Limitations:** The WOW runtime engine parses the supplied SQL to enable prompting and other parameter support. Due to varying levels of SQL support and features from database vendors. WOW only supports a limited subset of SQL features.

### UDF Support: User Defined Functions

WOW parses the SQL for variable substitution. To identify a SQL segment as a UDF you must name UDFs like: myLogic\_UDF

```
IE: select * from x.y where my_UDF(aFld) > 77
```

In this case, "my\_UDF" is a function created by the user, we will require WOW users to append "\_UDF" in the UDF name.

## HTML Code Operations

Like the SQL Operations described above, HTML operations use HTML to create operations that display HTML code. The HTML Operations can create a new level of customization by using HTML to enhance WOW Applications. Below is an example of an HTML Operation being used to create a welcome screen for a WOW application.



## Other Operation Types

### Operation Types

- **Associated Java Operation** - [PRO] Specifies a Java class where actual code can be executed.
- **Associated Join** - [PRO] An operation that joins data from two separate systems.
- **Association 1-1** - An operation associated with a field. This operation will display a single row.
- **Association 1-Many** - An operation associated with a field. This operation will display a results table (one or more rows).
- **Authentication** - Authentication allows for added security when using a WOW Application. Each user will have to enter a username and password before they can view or edit the application in question.
- **Auto Populate** - [EE] An SQL operation associated with a field that will retrieve information and populate other fields in the given Row based on the value of that field.
- **Auto-run - Batch Process** - [PRO] An operation that is scheduled to run automatically when an application is started.
- **Auto-run - Email** - [PRO] An SQL operation that returns a list of rows which contain email fields (usage ID -40). These email fields can be used in conjunction with the Auto Run capabilities of WOW.
- **Blob Download** - [PRO] An operation associated with a field (similar to *Association 1-1*) where Blob data (.jpg, image, etc.) is downloaded when the field is clicked on.
- **Execution Group** - Operation that actually runs one or more other operations. After defining the *Execution Group* operation, define other (normal) operations and set their "parent operation" to the *Execution Group* operation. The other operations will not appear in the TOC, but instead will be run when the *Execution Group* operation is run. For example, if the other operations were called OP1 and OP2, when the *Execution Group* operation is run, the results would contain results from OP1, followed by results from OP2.
- **File Upload** - [PRO] Operation associated with a field (similar to *Association 1-1*) where a file is uploaded when a field is clicked.
- **HTML Code** - Inserts HTML directly into your applications. This can be used to customize your program with a startup screen, logo, or any other custom HTML code you would like to add to your application.
- **HTML Code Association** - As the name suggests, this is the association version of the HTML Code operation. It provides the same functionality as the HTML Code operation but is set on a field as an association. However, since it is an association, row parameters (??field) can be placed anywhere in the HTML code.
- **HTML Reference** - Specifies a Web Site address that WOW will open in a new window.
- **HTML Reference Association** - [PRO] Similar to all associations in that upon execution, values from a source row may be used to retrieve dynamic content for an http URL. For example, if a selected record contained address information, you could create an *HTML Reference Association* that linked to Google™ Maps or some other Internet mapping service. Dynamic content in the URL can be replaced in a similar fashion to replacing values for SQL fields.
- **JSP Reference** - [PRO] Inserts a JSP file directly into your applications. Use the *JSP File* field, to specify the path to the JSP file.
- **Possible Values** - Uses data from a database to create the possible values of the field.



- **Possible Values Search** - *Possible Values* operation opens in a popup window so that the *Possible Value* for a particular field can be picked from a result set dynamically. Allows the user to specify search parameters and see other values in the row while selecting field value
- **Possible Values Selector** - Behaves in a similar fashion to the *Possible Values* operation; however, this operation causes a round-trip to the server when the value of the field changes.
- **Referrer Authorization** - Used to allow only users coming from a certain web page into your application.
- **Tabbed** - A secondary operation displayed in a tabbed layout.
- **User Authentication List** - An application can be secured by creating a *User Authentication List* operation which is defined by a comma separated list of user names and passwords. When the user logs on to an application with list based security they are prompted for their user name and password. This is a useful option when the WOW developer wants to quickly implement application level security for a small group of users without having set up table or user profile based security.
- **User Authorization List** - An operation that holds a static list of user names (in the operation's code field) defined when this operation is created and is used to restrict access to certain fields or operations. This authorization operation is useful when a small number of users will have restricted access to certain fields and/or operations.
- **User Authorization Operation** - An operation that dynamically returns a single column of user names. This type of operation is a more dynamic solution than the user authorization list. An SQL statement is defined that returns a result containing a single column of user names.
- **User Group Authorization List** - [EE] An operation that holds a static list of user group names (in the operation's code field) defined when this operation is created and is used to restrict access to certain fields or operations. Users can be designated to belong one or more groups (e.g. LDAP groups). This authorization operation is useful when a small number of users will have restricted access to certain fields and/or operations. The application signon would also need to be configured to collect a user's list of groups they belong to. Currently, only LDAP groups are supported (see [Configure Group Search Properties](#) for more details). [Minimum Version: WOW 7.0]
- **User Group Authorization Operation** - [EE] An operation that dynamically returns a single column of user group names. This type of operation is a more dynamic solution than the user group authorization list. An SQL statement is defined that returns a result containing a single column of user group names. The application signon would also need to be configured to collect a user's list of groups they belong to. Currently, only LDAP groups are supported (see [Configure Group Search Properties](#) for more details). [Minimum Version: WOW 7.0]
- **View Selected Association** - An operation associated with a field. This operation will display the selected record (source row) in a *Details* view without hitting the database again.

## Controlling the Display Order of Operations

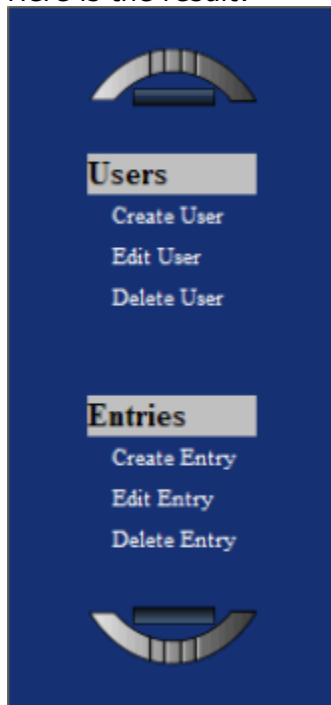
Display Groups determine how WOW separates different operations. All operations with the same Display Group will be grouped together when the application is run. Any name that would help with grouping the operations will work in this field. If the Display Group is not specified, that operation will appear under the *Default* group.

The parameter that specifies which Display Group that an operation appears in is the Display Group parameter. There are two elements involved in the ordering of operations. One is the order that the Display Groups appear. The other is the order in which the operations appear within the Display Group. Both of these order elements are determined by the single property Display Order. Within a Display Group, the operation with the lowest value appears first in the list. The next greater value in that Display Group appears second, etc. The order of the Display Groups is determined by the operation in each group with the lowest Display Order. The minimum value in each Display Group is then used to place the Display Group in the proper order.

For example, if we have the following operations and their associated properties, the operations will be displayed as shown:

Operation	Display Group	Display Order
Create User	Users	0
Edit User	Users	10
Delete User	Users	20
Create Entry	Entries	100
Edit Entry	Entries	110
Delete Entry	Entries	120

Here is the result:



Swapping the display order values of the *Create User* and *Delete User* operations, will in

turn swap the order of the operations within the *User Display Group*.

Operation	Display Group	Display Order
Create User	Users	20
Edit User	Users	10
Delete User	Users	0
Create Entry	Entries	100
Edit Entry	Entries	110
Delete Entry	Entries	120

Changing the display order of the *Create Entry* operation to a value of -100 will cause the minimum value of the display order for the *Entries* display group to be less than the *Users* display group, which will switch the order of the entire set. Despite the fact that the *Edit Entry* operation and the *Delete Entry* operation both have a display order value that is greater than all of the operations in the *Users* display group, the *Entries* display group is displayed first

Operation	Display Group	Display Order
Create User	Users	20
Edit User	Users	10
Delete User	Users	0
Create Entry	Entries	-100
Edit Entry	Entries	110
Delete Entry	Entries	120

Here is the result:



## Setting a Next Operation

Also known as work flow, WOW provides the ability to specify the next operation to be executed after a user inserts, updates, or deletes a record. For instance, let's say you have a multi-stage registration process with each stage handled by a separate operation. Rather than having the user select each stage individually from the menus, you could use the Next Operation feature to automatically direct the user from one operation to the next.

To direct the user from the Stage 1 operation to the Stage 2 operation, you would need to edit the Stage 1 operation, go to the *Advanced* section, and specify Stage 2 as the Next Operation.

**Advanced**

Connection Alias	-- None --	Operation Class	
Row Count	50	Row Coll. Class	
Row Class		Parameters JSP	
Caching Level*	Check cache and cache results	JSP File	<input checked="" type="radio"/> -- None -- <input type="radio"/>
Details JSP		Parent Operation	-- None --
Depends On	-- None --	Usage Id	<input checked="" type="radio"/> -- None -- <input type="radio"/>
Execution Rule	-- None --	Next Operation	Stage 2

## Using a Please Wait Page

The amount of time it takes for a query to run depends on several factors, including the complexity of the operation, the amount of data being searched over, and the speed of the database connection. WOW allows you to display a "please wait" page for long running queries. This please wait page appears immediately when a long-running query begins, and is replaced with the query results upon the completion of the operation. Showing a please wait page gives the user a more responsive experience than an application which appears to do nothing for several seconds after a search is initiated.

To indicate that a query should display a please wait page to the user while it runs, you must add the PleaseWait property group to the operation's Properties:

```
PleaseWait {}
```

You do not need to include any properties in the property group; the group itself is sufficient. The operation will now display the default please wait page to the user when it runs.

## Creating a Custom Please Wait Page

[PRO] If you do not want to use the WOW default please wait page, you can create your own custom page to use instead. You can use any valid JSP as your please wait page. Any HTML page will work as long as you change the file extension from .html to .jsp or you can create a custom JSP using scriptlets and tag libraries. Keep in mind that WOW will not add any sort of header or menus to your please wait page.

Inside of the PleaseWait property group, you use the JSP property to indicate which JSP should be used as the please wait page:

```
PleaseWait { jsp: /mydir/mysubdir/mypleasewait.jsp; }
```

The operation will now use your please wait JSP instead of the default WOW page.

**NOTE:** The JSP file path used in the example above is relative to the WOW context folder in Apache Tomcat. For instance, the file path above would be pointing to the following address: ...\\Tomcat 5.5\\webapps\\wow65\\mydir\\mysubdir\\mypleasewait.jsp

If multiple operations are using the same please wait JSP, you can specify the PleaseWait property group containing your JSP in the application properties. Then all operations with please wait pages will use the JSP in the application properties by default. (You must still specify the PleaseWait property group in the operation to indicate that the operation should show a please wait page, but you do not have to include your custom JSP in the operation properties.)

# Selecting Records

## Basic SQL Queries Using the SELECT Statement

The SELECT statement is a vital SQL command which is used frequently within WOW. This section contains numerous examples in which some basic SQL knowledge will be helpful in understanding. In all of the examples we will only need to complete the fields located in the BASIC section of an operation. For each example there will be a screen shot followed by an explanation of its contents.

The screenshot shows a form titled "Basic" with the following fields:

- Label\*:** Employees
- Type\*:** SQL (dropdown menu)
- Title:** Sample Employees
- Description:** View a sample database of employees
- Operation Code:** SELECT \* FROM SAMPLE.EMPLOYEE
- Instructions:** (empty text box)
- Application:** [View Application](#)

To get to this screen, click on the *Insert Operation* button while viewing a list of operations. The focus will be on the SQL query. SQL is not case sensitive, but entering SQL in all capital letters will simplify the coding. Take for example this simple SQL query:

```
SELECT * FROM SAMPLE.EMPLOYEES
```

This query will select all of the records in the EMPLOYEES table, which is found in the SAMPLE schema. The \* symbol is used to collect all of the columns from the table. Most SQL queries will have the keyword FROM within the query statement. This tells the program which table(s) to select the information from. After you have entered all of the relevant data click the *Insert Operation* button. This will insert the SQL operation into the application.

**NOTE:** The SELECT statement syntax is slightly different for Microsoft's SQL Server and Access.

### Microsoft SQL Server 2000 & Up

With SQL Server, the database schema notation is slightly different than, let's say, the AS/400. In the examples below, two different and valid notations are given. In the first, note the use of two periods between the database and table. In the second statement, notice the inclusion of OWNER between the database and table.

```
SELECT * FROM DATABASE..TABLE (recommended)
```

or

```
SELECT * FROM DATABASE.OWNER.TABLE
```

### Microsoft Access

```
SELECT * FROM TABLE
```

## Other Queries Using the SELECT Statement

SQL allows you to select specific columns from a table.

<b>Basic</b>			
Label*:	<input type="text" value="Employees"/>	Title:	<input type="text" value="Sample Employees"/>
Type*:	<input type="text" value="SQL"/>	Description:	<input type="text" value="View a sample database of employ"/>
Operation Code:	<input type="text" value="SELECT FIRSTNME, LASTNAME, HIREDATE FROM PJDATA.EMPLOYEE"/>		
Instructions:	<input type="text"/>	Application:	<a href="#">View Application</a>

In this example, the SQL statement is:

```
SELECT FIRSTNAME, LASTNAME, HIREDATE FROM PJDATA.EMPLOYEE
```

This SQL query is selecting individual columns in the EMPLOYEE table. LASTNAME, FIRSTNAME and HIREDATE are the names of specific columns. Selecting specific columns will prevent you from displaying data that isn't relevant to the search. Each column name is separated by a comma. Again notice the FROM keyword pointing to the EMPLOYEE table. After you have entered all of the relevant data, click the Insert Operation button. This will insert the SQL Operation into the application.

The next example shows how to use the WHERE clause in an SQL statement to restrict the rows returned by the query. Only those rows meeting the criteria in the WHERE clause will be returned by the query.

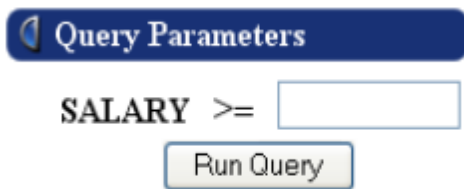
<input type="button" value="Insert"/> <input type="button" value="Cancel"/>			
<b>Basic</b>			
Label*:	<input type="text" value="Employees"/>	Title:	<input type="text" value="Sample Employees"/>
Type*:	<input type="text" value="SQL"/>	Description:	<input type="text" value="View a sample database of emplo"/>
Operation Code:	<input type="text" value="SELECT * FROM PJDATA.EMPLOYEE WHERE SALARY &gt;= ?"/>		
Instructions:	<input type="text"/>	Application:	<a href="#">View Application</a>

The SQL statement used in this example is:

```
SELECT * FROM PJDATA.EMPLOYEE WHERE SALARY >= ?
```

This SQL query is different from the previous two because of the WHERE keyword that is added. The WHERE clause is used to specify a search condition that will identify the row or rows you want to manipulate. Notice that the WHERE clause contains a question mark. If we

knew what value we wanted entered ahead of time, we could hard code the value into the SQL Operation. A question mark allows the user to provide any value at runtime.



A screenshot of a web form titled "Query Parameters" in a dark blue header. Below the header, the text "SALARY >=" is displayed next to a white text input field. Below the input field is a light gray button with the text "Run Query".

After you enter the value for SALARY, the query will display a table with all of the employees with salaries greater than or equal to the value specified. The WHERE clause is usually used with comparison operators. You can include multiple parameters in the WHERE clause to specify more complex queries.



## Setting Key Position Field to Select Unique Records

In WOW many of the SQL statements are generated for the user. For example when selecting records from a field and then clicking insert, the SQL is dynamically generated. If there are many records, you may come across a problem where WOW will Select, Delete, Insert, Update more than the one record desired because it does not have the appropriate key position(s) set.

The key position is the same as a primary key in a database system and similar to a unique key. It gives the database a reference to differentiate rows (records). Normally, if a primary key is set in database system WOW will pick it up. If not, or if the key position(s) was not set in the database, then it is necessary to set the Key Position in the Field Descriptor to differentiate the rows.

Database Settings	
Library Name * : <input type="text" value="WOWSAMP60"/>	Table Name * : <input type="text" value="PRODUCTINV"/>
	System Alias * : <input type="text" value="SAMPCONN"/>
SQL Type * : <input type="text" value="CHAR"/> ▼	SQL Type Name * : <input type="text" value="CHAR"/>
Column Size: <input type="text" value="10"/>	Scale: <input type="text" value="0"/>
Nullable: <input type="text" value="No Nulls"/> ▼	Key Position: <input type="text" value="1"/>

If there is no specific field such as ID or employee number use multiple fields that guarantee unique records. In this case set key position for different fields to 1,2,3...etc.

**NOTE:** These key fields do not need to be shown. They just need to be set. You can set them to not display by changing the display property group of the operation.

# SQL Tips

## Case Sensitivity

To make SQL searches not case sensitive use the UPPER keyword. For example:

```
SELECT * FROM yourtable WHERE UPPER(lname) LIKE UPPER(TRIM(CAST(? as CHAR(20)
)))
```

The field name was a CHAR(20) and we wanted to search for it using any number of known letters in the word. We need to use a TRIM command, because, for example, if you search for all last names starting with an 'S', it will take the 'S' and append zeros to fill up the CHAR(20) space unless it matches it exactly. So, the TRIM takes out these extra spaces and fillers.

## Optional Values

To allow optional entry of certain values in a SQL search statement, use the SQL VALUE clause. For example:

```
SELECT * FROM yourtable WHERE firstname = VALUE(CAST(? as CHAR(20)),
firstname) AND lastname = VALUE(CAST(? as CHAR(20)), lastname)
```

In this case, you are searching your table by last name and first name using the VALUE function which basically is a function that returns the first value that is not null. For example, we could input a first name and last name, one separate from the other or none at all and it will search according to that. If you enter none of these, it will then show the entire file. If you only enter the first name, it will search for all of the records that have the same first name and will not use the last name field as a parameter for searching. This allows you to have one search operation that can have many different fields to search by without depending on each other on having a value.

**NOTE:** You may also use the LIKE function with the VALUE function to make the search even more powerful; however, with LIKE, you again may need to use the TRIM command.

# Inserting Records

## Basic SQL Queries Using the INSERT Command

[EE] This section will cover basic INSERT commands using SQL statements and will include examples to help with the explanation. Once again, a basic knowledge of SQL is recommended.

Basic			
Label*:	<input type="text" value="Add Employee"/>	Title:	<input type="text"/>
Type*:	<input type="text" value="SQL"/>	Description:	<input type="text" value="Add a new Employee"/>
Operation Code:	<input type="text" value="insert into pjdata.employee"/>		
Instructions:	<input type="text"/>	Application:	<a href="#">View Application</a>

The insert command is followed by the INTO keyword which specifies the table to insert data into. The sample below is what will be displayed if the SQL operation is executed.

Fields marked with an asterisk (\*) are required.

				Insert	Cancel
EMPNO:	<input type="text"/>	FIRSTNAME:	<input type="text"/>		
MIDINIT:	<input type="text"/>	LASTNAME:	<input type="text"/>		
Dept:	<input type="text" value="(None)"/>	PHONENO:	<input type="text"/>		
HIREDATE (MM/dd/yyyy):	<input type="text"/>	JOB:	<input type="text"/>		
EDLEVEL:	<input type="text" value="0"/>	SEX:	<input type="text" value="(None)"/>		
BIRTHDATE (MM/dd/yyyy):	<input type="text"/>	SALARY:	<input type="text"/>		
BONUS:	<input type="text"/>	COMM:	<input type="text"/>		

				Insert	Cancel
--	--	--	--	--------	--------





After the values have been entered, click the *Insert* button. This will create a new record for the table. Only the required fields (indicated by the red asterisk) need to be given a value in

order for the row to be inserted.

## Inserting Records without SQL Commands

WOW makes it possible to insert rows without an INSERT SQL operation. If you specify on an SQL operation to "Allow Inserts", WOW provides the insert functionality for you. For example, a SELECT statement will result in a screen with a table of results that has an *Insert* button below it. To insert a new row, simply press the *Insert* button. After the *Insert* button has been clicked the follow screen appears:

Fields marked with an asterisk (\*) are required.

		<input type="button" value="Insert"/>	<input type="button" value="Cancel"/>
EMPNO:	<input type="text"/>	FIRSTNAME:	<input type="text"/>
MIDINIT:	<input type="text"/>	LASTNAME:	<input type="text"/>
Dept:	(None) 	PHONENO:	<input type="text"/>
HIREDATE (MM/dd/yyyy):	<input type="text"/> 	JOB:	<input type="text"/>
EDLEVEL:	<input type="text" value="0"/>	SEX:	(None) 
BIRTHDATE (MM/dd/yyyy):	<input type="text"/> 	SALARY:	<input type="text"/>
BONUS:	<input type="text"/>	COMM:	<input type="text"/>
		<input type="button" value="Insert"/>	<input type="button" value="Cancel"/>

After filling in all of the information that is pertinent, click the *Insert* button. This will give you a message letting you know that your row has been inserted into the database.

## Inserting Records Using Parsing

[EE] When inserting a record using parsing, you must supply a value for each column to add data to. A value in the VALUES clause for each column named in the INSERT command's column list must be provided. If a column has a default value, the keyword DEFAULT may be used as a value on the VALUES clause. The image below shows a sample SQL INSERT command using parsing.

**Basic**

Label:  Title:

Type:  Description:

Operation Code:

Instructions:  Application: [View Application](#)

Each column is separated by commas. Next is the values keyword that is followed by the three values to be added to the table. The values in the second set of parenthesis should correspond with the column names in the first set of parenthesis. Run Time prompting ('?') can be used in place of any hard coded value. Running this INSERT command will only display the values you choose in the INSERT command.

Last Name:  WORKDEPT:

Salary:

Only the three fields and their values that were specified in the INSERT command will be added. Using parsing allows users to only enter the information they want added to the record. In the event that there is a required field in the table into which you are inserting records, you will have to be sure to give the required field a value or else an error will occur.

## Joined Inserts

[EE] A joined insert is an insert statement which inserts data into multiple database tables. For example, the following operation inserts data into two tables, the CUSTOMER and CUSTLOC tables.

The screenshot shows a 'Basic' tab with the following fields:

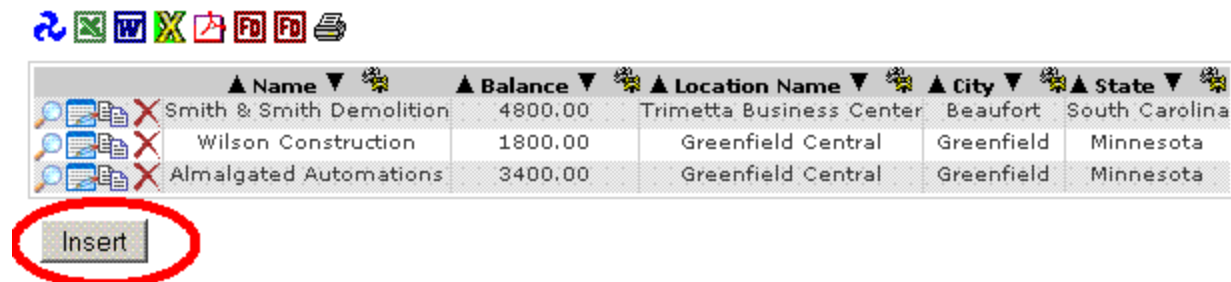
- Label:** Customers Insert
- Title:** (empty)
- Operation Type:** SQL (selected from a dropdown menu)
- Description:** (empty)

The SQL statement entered in the description field is:

```
INSERT INTO JETEMP.CUSTOMER, JETEMP.CUSTLOC  
(CID, CNAME, CBALANCE, CLOCID, LOCID, LOCNAME, LOCCITY, LOSTATE)  
VALUES (?, ?, ?, ?, ?, ?, ?, ?)
```

**NOTE:** Field names beginning with "C" are from the CUSTOMER table, and fields beginning with "L" are from the CUSTLOC table

Joined inserts can also occur when a join is used to select multiple rows from the database (with a query like `SELECT * FROM JETEMP.CUSTOMER JOIN JETEMP.CUSTLOC ON CLOCID = LOCID`), and the user clicks the corresponding Insert button.



During a joined insert, the user is shown fields from all tables on the same insert screen. There is no indication given to the user of which fields belong to which tables.

<input type="button" value="Insert"/>	
<b>Customer ID</b> <input type="text"/>	<b>Location ID</b> <input type="text"/>
<b>Name</b> <input type="text"/>	<b>Location Name</b> <input type="text"/>
<b>Balance</b> <input type="text"/>	<b>City</b> <input type="text"/>
<b>Location ID</b> <input type="text"/>	<b>State</b> <input type="text" value="- Choose -"/>

<input type="button" value="Insert"/>
---------------------------------------

Additional properties can be used to control join behavior. [See the Join property group](#):

## Restrictions

Each field name in a joined insert must apply to a single table. When two or more tables have columns with the same name, those columns cannot be used in a joined insert into those tables.

## Transactions

[EE] The SQL standard does not allow for inserting into multiple tables with a single statement, therefore internally WOW splits the insert statement into multiple separate SQL statements and sends them all to the database. It is possible for the one statement to succeed but for the other statements to fail (a dropped network connection or an authorization error are two things that could cause one statement to work and then others to fail). When the first statement fails, WOW abandons the insert and reports the error as usual. However, if a subsequent statement fails after the first statement succeeds, the database could be left in a corrupted state. For this reason you may wish to configure your application not to use joined inserts, or to use database transactions.

A transaction is a way of bundling multiple SQL statements into a single unit of work – that unit of work will either succeed or fail as a whole. If it fails then none of the statements in the transaction will have affected the database. Some databases do not support transactions, and other databases require special configuration before transactions can be used. Check with your database documentation to find out how to configure transactions on your database.

By default, WOW will not use transactions for joined inserts. If you want WOW to issue your joined insert as a single transaction (which is recommended if your database supports transactions), you must use the Join property group to specify this:

```
Join { transactions: true; }
```

This property group should be placed in the properties field of the operation which is inserting the joined rows into the database (or the operation which selected the joined rows, depending on which operation is being run). Alternatively you can place this property group in the properties field of the application, where it will apply to all of the operations in that application.





# Updating Records

## Basic SQL Queries Using the UPDATE Command

[EE] The UPDATE statement is used to change data in a table. Using the UPDATE statement allows you to change the value of one or more columns in each row of the table. The screen shot below shows an example of an UPDATE statement. Previous knowledge of SQL and familiarity with the UPDATE statement is recommended.

Basic			
Label *	<input type="text" value="Update Bonus \$200"/>	Title:	<input type="text"/>
Type *	<input type="text" value="SQL"/>	Description:	<input type="text" value="Update Bonus \$200"/>
Operation Code:	<input type="text" value="update pjdata.employee set bonus = bonus + 200"/>		
Instructions:	<input type="text"/>	Application:	<a href="#">View Application</a>

The SQL statement in the *Operation Code*:

```
UPDATE PJDATA.EMPLOYEE SET BONUS = BONUS + 200
```

This is a simple SQL query that is updating (giving a raise to) all employees. The basic syntax of an UPDATE statement is listed above. The UPDATE keyword is directly followed by the name of the table to be updated. In this case it's the EMPLOYEE table in the PJDATA schema. The SET clause names the columns you want to be updated and provides a value for you to update. The bonus for all employees is 200 dollars. The new value for bonus will then be the old value plus 200 more.

## Using a WHERE clause with the UPDATE statement

The WHERE clause and UPDATE statement are commonly used together. By using the WHERE clause in conjunction with the UPDATE statement, a user can specify only certain values that meet a certain criteria. The screen shot below will show how to use the WHERE clause in an UPDATE query.

<b>Basic</b>			
Label* :	<input type="text" value="Update Salary by Dept No"/>	Title:	<input type="text"/>
Type* :	<input type="text" value="SQL"/>	Description:	<input type="text" value="Update Salary by Dept No"/>
Operation Code:	<input type="text" value="UPDATE.PJDATA.EMPLOYEE SET SALARY = SALARY + 1000 WHERE WORKDEPT=?"/>		
Instructions:	<input type="text"/>	Application:	<a href="#">View Application</a>

The SQL statement in the SQL field:

```
UPDATE.PJDATA.EMPLOYEE SET SALARY = SALARY + 1000 WHERE WORKDEPT=?
```

The above statement is a simple SQL UPDATE query. The UPDATE command points to the location of the table. In this example the table is EMPLOYEE in the PJDATA schema. The SET command sets the salary to equal to the current salary and adds 1000. By using the WHERE statement the user can enter a specific work department; only employees in that department will receive raises. The question mark allows the user to enter any work department when this command is run.

## Joined Updates

[EE] A joined update occurs when a user edits and updates values in a joined row. A joined row is a row containing data from multiple tables. For example, a query like `SELECT * FROM JETEMP.CUSTOMER JOIN JETEMP.CUSTLOC ON CLOCID = LOCID` would result in joined rows.



	▲ Name ▼	▲ Balance ▼	▲ Location Name ▼	▲ City ▼	▲ State ▼
	Smith & Smith Demolition	4800.00	Trimetta Business Center	Beaufort	South Carolina
	Wilson Construction	1800.00	Greenfield Central	Greenfield	Minnesota
	Almalgated Automations	3400.00	Greenfield Central	Greenfield	Minnesota

Insert

During a joined update the user is shown fields from all tables on the same screen. There is no indication given to the user of which fields belong to which tables.

Previous		Update and Previous		Update		Cancel	
Name	Almalgated Automations		Location Name	Greenfield Central			
Balance	3400.00		City	Greenfield			
State	Minnesota						
Previous		Update and Previous		Update		Cancel	

Additional properties can be used to control join behavior. [See the Join property group:](#)

### Restrictions

Each field name in a joined update must apply to a single table. When two or more tables have columns with the same name, those columns cannot be used in a joined update to those tables. In addition, the row being used for the joined update must contain all the key fields for all of the tables being jointly updated.

### Transactions

The SQL standard does not allow for updating multiple tables with a single statement, therefore internally WOW splits the update statement into multiple separate SQL statements and sends them all to the database. It is possible for one statement to succeed but for the other statements to fail (a dropped network connection or an authorization error are two things that could cause one statement to work and then others to fail). When the first statement fails, WOW abandons the update and reports the error as usual. However, if a subsequent statement fails after the first statement succeeds, the database could be left in a corrupted state. For this reason you may wish to configure your application not to use joined updates, or to use database transactions.

A transaction is a way of bundling multiple SQL statements into a single unit of work – that unit of work will either succeed or fail as a whole. If it fails then none of the statements in the transaction will have affected the database. Some databases do not support transactions, and other databases require special configuration before transactions can be used. Check with your database documentation to find out how to configure transactions on your database.

By default WOW will not use transactions for joined updates. If you want WOW to issue your joined update as a single transaction (which is recommended if your database supports transactions), you must use the Join property group to specify this:

```
Join { transactions: true; }
```

This property group should be placed in the properties field of the operation which selected the joined rows. Alternatively you can place this property group in the properties field of the application, where it will apply to all of the operations in that application.

# Deleting Records

## Basic SQL Queries Using the DELETE Command

[EE] The DELETE statement is used to remove entire rows from a table. The DELETE statement cannot remove specific columns from a row. If the WHERE statement in a DELETE query is omitted, SQL will remove all the data in the table.

Basic			
Label* :	<input type="text" value="Delete Records by Ed. Level"/>	Title:	<input type="text"/>
Type* :	<input type="text" value="SQL"/>	Description:	<input type="text" value="Delete Records by Ed Level"/>
Operation Code:	<input type="text" value="DELETE FROM PJDATA.EMPLOYEE WHERE EDLEVEL &lt; ?"/>		
Instructions:	<input type="text"/>	Application:	<a href="#">View Application</a>

A simple SQL DELETE statement goes as follows:

```
DELETE FROM PJDATA.EMPLOYEE WHERE EDLEVEL < ?
```

Notice the keyword FROM following the DELETE command which specifies which table to delete data from. In this example, the SQL statement is deleting data from the EMPLOYEE table in the PJDATA schema. The WHERE clause is extremely important when using the DELETE command. All rows whose EDLEVEL is less than the value entered at runtime will be deleted from the table.

## Deleting Rows Without SQL Commands

[EE] It is possible to delete rows without an SQL Command. This is done by selecting specific rows from a table using their check boxes, and then clicking the Delete button under the data. The selected rows will be removed from the table. If you have set the SelectionType to none in the properties section this option is not available. Any of the rows could be deleted using this deletion method.

### Sample Employees



	▲ EMPNO ▼	▲ FIRSTNAME ▼	▲ MIDINIT ▼	▲ LASTNAME ▼
<input type="checkbox"/>	000010	CHRISTINE	I	HAAS
<input type="checkbox"/>	000110	VICENZO	G	LUCCHESI
<input type="checkbox"/>	000120	SEAN		O'CONNELL
<input type="checkbox"/>	200010	DIANE	J	HEMMINGER
<input type="checkbox"/>	200120	GREG		ORLANDO
<input type="checkbox"/>		Tony		Xerex
<input type="checkbox"/>	99	Bob		Kain
<input type="checkbox"/>	181	Paul	B	Holm
<input type="checkbox"/>	345567	ROSS		LARKEN

The checkboxes on the far left allows users to select individual rows. You would use the checkbox to select a row and then delete it.

## Joined Deletes

[EE] A joined delete occurs when a user deletes a joined row. A joined row is a row containing data from multiple tables. For example, a query like `SELECT * FROM JETEMP.CUSTOMER JOIN JETEMP.CUSTLOC ON CLOCID = LOCID` would result in joined rows.



	▲ Name ▼	▲ Balance ▼	▲ Location Name ▼	▲ City ▼	▲ State ▼
	Smith & Smith Demolition	4800.00	Trimetta Business Center	Beaufort	South Carolina
	Wilson Construction	1800.00	Greenfield Central	Greenfield	Minnesota
	Almalgated Automations	3400.00	Greenfield Central	Greenfield	Minnesota

Insert

Additional properties can be used to control join behavior. [See the Join property group](#):

## Restrictions

A joined row must contain all key fields for all of its tables – otherwise that row cannot be used for a joined delete.

## Transactions

The SQL standard does not allow for deleting records from multiple tables with a single statement, therefore internally WOW splits the delete statement into multiple separate SQL statements and sends them all to the database. It is possible for the one statement to succeed but for the other statements to fail (a dropped network connection or an authorization error are two things that could cause one statement to work and then others to fail). When the first statement fails, WOW abandons the update and reports the error as usual. However, if a subsequent statement fails after the first statement succeeds, the database could be left in a corrupted state. For this reason you may wish to configure your application not to use joined deletes, or to use database transactions.

A transaction is a way of bundling multiple SQL statements into a single unit of work – that unit of work will either succeed or fail as a whole. If it fails then none of the statements in the transaction will have affected the database. Some databases do not support transactions, and other databases require special configuration before transactions can be used. Check with your database documentation to find out how to configure transactions on your database.

By default WOW will not use transactions for joined deletes. If you want WOW to issue your joined delete as a single transaction (which is recommended if your database supports transactions), you must use the Join property group to specify this:

```
Join { transactions: true; }
```

This property group should be placed in the properties field of the operation which selected the joined rows. Alternatively you can place this property group in the properties field of the application, where it will apply to all of the operations in that application.



# Field Descriptors

Field Descriptors are an important and powerful part of WOW. A Field Descriptor describes a field within a table or database. Field Descriptors contain information such as the external name of a field, whether or not the field is required, and the type of data the field contains, such as numeric, time, etc. Utilizing Field Descriptors allows WOW to perform validation, display formatting, and other tasks for your application without any coding on your part. Field Descriptors should be created for all tables that WOW uses frequently.

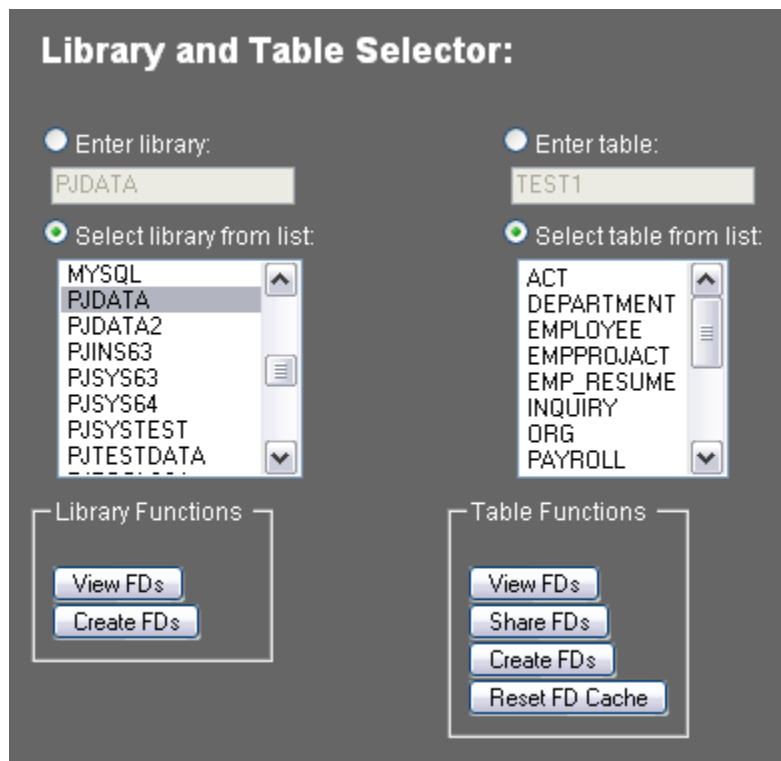
We encourage creating Field Descriptors for all production level tables/environments.

## Field Descriptor Manager

To edit or create Field Descriptors you first must be in the Field Descriptor Manager. The easiest way to get to the Field Descriptor Manager is to click on the small "FD" icon above any results table.



This will bring up the Field Descriptor Manager in a new window:



You can also start up Field Descriptor Manager from WOW directly without first running your application. Click on the Setup Connections link on the left side of the builder. You can then select a connection whose field descriptors you want to manage and click the Edit FDs field. **Whenever you are using Field Descriptor Manager, you are managing the field descriptors for a single connection.** To manage field descriptors for another connection, you must choose a different connection (or a table retrieved using that connection) and open up Field Descriptor Manager from that connection or table.

The "Enter library" and "Enter table" fields at the top of the main Field Descriptor Manager screen allow you to type in the name of the library and table whose Field Descriptors you want to work with or create. Alternatively, you can click one of the "Select from list" radio

buttons and choose the library or table from the list of libraries or tables available (double click an entry in the list to select it) Once you have picked a library or table to work with, the following options are available:

### Library Functions

- **Enter Library** - This shows what library you are currently in. If you would like to change your current library click the radio button below and select the library in the list given. If a table's FDs are shared, specify \* for the library name.
- **View FDs** - Displays all of the Field Descriptors within the entire library. This may take some time to complete, depending on the number of tables in the library. It is usually better to display the Field Descriptors for a single table at a time.
- **Create FDs** - Creates Field Descriptors for the entire library. Creating Field Descriptors for the entire library is not recommended – it is usually better to create them separately for each table.

### Table Functions

- **Enter Table** - This shows you which table you are currently in. If you would like to switch your current table, click the radio button below and select the table you would like in the list given.
- **View FDs** - Displays all of the Field Descriptors already created for the table. In most cases the FDs are already displayed below, if for some reason the FDs are not displayed click this button and they will be displayed.
- **Share FDs** - Allows you to share the Field Descriptors for this table with other tables which have the same name, but are in different libraries. By default, a table's Field Descriptors are only for that table, and not for any other tables, even if those tables have the same name. This function will not share Field Descriptors for multiple connections. Each connection holds a unique set of Field Descriptors. If a table's FDs are shared, you must specify \* for the library name.
- **Create FDs** - Creates field descriptors for the currently selected table.
- **Reset FD Cache** - For advanced users only. This button is used for troubleshooting purposes. Internally, WOW caches Field Descriptors as they are read – this button clears the cached FDs. This function can be helpful when FD changes recently made do not appear to be affecting the operations using this table and connection.

## Editing Rows Within the Field Descriptor Manager

To change the settings on an individual row that is shown under the Field Descriptor section, click the edit icon next to the Field Descriptor you wish to edit. This will bring up a screen containing the Field Descriptor settings that can be edited. The screen shot below covers the first two sections *Basic Settings* and *Display Settings*.

The screenshot displays a web-based configuration interface for a Field Descriptor. It is divided into two main sections: 'Basic Settings' and 'Display Settings', each with a red header bar and a plus icon. The 'Basic Settings' section contains fields for 'Field Name\*' (ID), 'External Name' (ID), 'Required' (checkbox), 'Required On Search' (checkbox), 'Default Value' (radio buttons with a dropdown set to '-- None --'), and 'Auto Update Value' (checkbox). The 'Display Settings' section contains fields for 'Field Set' (radio buttons), 'Display Order' (text input with '0'), 'Display Rule\*' (dropdown set to 'Never'), 'Display Component\*' (dropdown set to '(Default)'), 'Help Text' (text input), 'Style Class' (text input), 'Display Width' (text input), and 'Display Height' (text input).

### Basic Settings

- **Field Name** - The database name of the column which this Field Descriptor describes. In the above example, the Field Descriptor for the SERVICE\_COMPANY column is being edited.
- **External Name** - The external name given to the field. External names are used to make the field name more user-friendly.
- **Required** - Indicates whether or not the described field is required. Required fields are denoted by a red asterisk, and are validated when a row is updated or inserted. If a required field is left blank on an insert or update, a validation error is displayed to the user.
- **Required on Search** - Indicates that if the field is used by a search parameter (ex. WHERE FLD1 = ?), a value must be specified before the search operation is performed.
- **Default Value** - The default value for the field. A default value can be any value you want initially displayed. If the field is auto-incremented, the default value can be used to set the starting value.
- **Auto Update Value** - [EE] This value will set for the field when an update is performed. The same rules apply to this setting as the Default Value setting. A

common use for this option would be for a field that is designated to contain the "last changed" date, time, or timestamp value. In this particular example, you would set this option's value to *\*current*. Any time this row is successfully updated, the last changed value would be updated as well. [WOW 7.1] WOW only includes fields whose values have changed when generating an SQL UPDATE. In some cases such as workflow applications, you may want a field to be updated into the database regardless if it changed. For example: `SELECT 'APPROVED' as status, B, C FROM LIB.REQUESTS ...` When updating, you may want STATUS included when the row is updated. Supports a special value of *\*VALUE* to indicate that this field should be included when an edit/update is performed.

## Display Settings

- **Field Set** - Field Sets are used to group different fields together. By entering a specific name in the Field Set section you can group different fields by the name given (*Basic Settings* and *Display Settings* are examples of Field Sets).
- **Display Order** - Used to display fields in order specified by the number given (in ascending order). You can use any numbers when ordering fields.
- **Display Rule** - Contains the rule to use when displaying this field. You can set the display rule to one of the following options:
  - Always - This field should be displayed whenever its row is displayed. This is the default setting.
  - Hide on insert - The field should not be displayed when its row is being inserted.
  - Hide on update - This field should not be displayed when its row is being inserted.
  - Hide on update and insert - [EE] This field should not be displayed when its row is being updated or inserted.
  - Never - This field should never be displayed.

There are other display rule options that are for WOW customer programmer only – you should only set the display rule to one of the values listed above.

- **Display Component** - Indicates which display component should be used to display the field. Most of the time, WOW will pick which graphical component to use when displaying a field based on the type of data contained in the field. If you want to override the system default (for example, by using a text area instead of a text field to display character data), you can choose which display component to use.
  - (Default) - Component is determined by the DataEngine.
  - \*Associated Operation - Rather than generating a normal association hyperlink, this runs the associated operation and returns the results within the actual field. Note, this option can only be selected if an Associated Operation has been specified in the Advanced Settings section.
  - EditableSelect - Two components are generated for this option: a top drop down menu with selectable possible values and a text field that allows entry for a new value.
  - EditableSelect - Text Area - Same as an EditableSelect except insures that the bottom component will be a text area instead of a text field.
  - List - Component is a selectable possible values list displaying all options in which multiple values can be selected.
  - RadioButtonPVSelect - Displays a possible value selection using radio buttons instead of selection list. **(For future support)**
  - TextArea - Uses a text area instead of a text field for the display component.
- **Help Text** - Text that will be displayed when the user positions their pointer over the component

- **Style Class** - The fully qualified name of a CSS class that will perform formatting on the look of the component generated for the field. For example, if you need to right justify a field, you can create a unique CSS style class and specify the class name here. You would create a `fieldstyles.css` file and define in it: `.mystyle { text-align: right; }` and set the Style Class to `mystyle`. If the field is rendered as an image (`<img ...>`), the class would need to be named as the following in CSS: `".mystyle img"`.
- **Display Width** - Sets the width of the display in pixels (changes with HTML settings).
- **Display Height** - Sets the height of the display in pixels (changes with HTML settings).

### Possible Value Settings

**Possible Value Settings**

Possible Values Key  ▼

Possible Values Operation -- None --

Possible Value Class ☒ -- None -- ▼

- **Possible Value Key** - The key that determines which Possible Values to use for this field.
- **Possible Value Operation** - This is a drop-down menu listing all of the previously created possible value operations.
- **Possible Value Class** - The specific Java class that is used to hold Possible Values internally on the system. You may also specify a Possible Value class name.
  - **\*DISTINCT\*** - Returns all distinct values from the database for that particular field. NOTE: Using this for a list of records will degrade performance. For multiple records use **\*DISTINCT-CACHE\***
  - **\*DISTINCT-CACHE\*** - Returns all distinct values from the cached contents of that particular field. **NOTE: For performance reasons, CACHE should be used whenever a list of records will be shown.**

### Advanced Settings

Advanced Settings

Field Class

-- None --

Field Descriptor Type

Derived

Getter Method

Association Operation

URL Association

Remarks

XML Tag

Formatter Class

-- None --

Concurrency

Concurrent

Setter Method

Notify Status Change

No

- **Field Class** - The specific Java class that will be used internally to store the data contained in the field. These classes usually have specialized formatting or display information built in that can be used for fields containing certain types of data. For example, when the `planetj.database.field.PasswordField` class is selected for a field descriptor, fields using that field descriptor will hide their content with asterisk characters. Users with WOW Professional Edition may create their own Field Subclasses. [PRO]
  - Field Class Parameters - If you manually enter the Field Class, you can optionally enter special parameters. These parameters indicate features or attributes of the Field Class. The following are Field Class parameters that you can enter: `minLength=xx` where `xx` is the minimum length of the field (ex. `minLength=7`). To enter a Field Class parameter, you must enter the full package and class name followed by a comma (,) followed by a `parameter=value` where *parameter* is the Field Class parameter. If there is more than one parameter, they should be separated by a comma (,). Example: `planetj.database.field.PasswordField,minLength=7,digitRequired=true` Some Field Classes have special Field Class parameters that are specified only to that field. For instance on a `DateField`, you can specify the date format. In the above example, you will notice the `digitRequired=true` Field Class parameter is used; that is a special parameter for the `PasswordField`. Here are the preset Field Classes:
    - **Address1** - Designates the field as the first line of a street address (for use elsewhere in the application). No specified validation is performed.
    - **Address2** - Designates the field as the second line of a street address (for use elsewhere in the application). No specified validation is performed.
    - **Area Code** - The validation of an area code field ensures that the length of the area code is the correct length and only contains numeric digits. If the values contains (, ), or -, they are also accounted for.
    - **Auto Update Timestamp** - This sets a new timestamp value when a record is being inserted or updated. Alternatively, the field's Default Value and Auto Update Value can be set to `*CURRENT`. This will achieve the same

effect on TimeStamp fields.

- **City** - Designates the field as a city (for use elsewhere in the application). No specific validation is performed.
- **Credit Card Expiration Month** - If no other Possible Value Key or Operation is specified, this Field Class uses the Possible Value Key `*MONTHS_OF_THE_YEAR*` to return a list of all months.
- **Credit Card Expiration Year** - If no other Possible Value Key or Operation is specified, this Field Class uses the Possible Value Key `*CC_EXPIRE_YEAR*`.
- **Credit Card Number** - The validation of a credit card number ensures that the length of the number is the correct length (16 or less digits).
- **Credit Card Type** - If no other Possible Value Key or Operation is specified, this Field Class uses the Possible Value Key `CC_TYPE` to return a list of standard credit card types (Visa, MasterCard, etc.).
  - a. All records in the last 6 months  
`SELECT...WHERE...DATE ((CENTURY * 100 + YEAR) || '-' || MONTH || '-' DAY) > CURRENT DATE - 6 months`
  - b. All records in the last 3 months  
`SELECT...WHERE...DATE ((CENTURY * 100 + YEAR) || '-' || MONTH || '-' DAY) > CURRENT DATE - 3 months`
- **Date** - In the database, a date field might not necessarily be a Date object. It could be a String or a number. To allow for proper reading and setting to the database, a Field's Field Class could be set to `DateField` followed by a comma and any user-defined pattern for date formatting. The pattern should be the format the value should be written as when inserted or updated to the database.

Example:

In the database, we have a field that is a CHAR with a length of 8. It takes the format 2 month, 2 day, and 4 year. When we read the value from the database, it needs to be modified from a string of characters into a date before it is displayed. If we set the field class to `DateField`, then a Java Date object will be generated, which gives the field's value more meaning and flexibility. In this case, we would set the field class to this value: `planetj.database.field.DateField,MMddyyyy` This is the fully qualified class name of the field descriptor followed by a comma and then the pattern that the date needs to be in order to be written to the database. This lets WOW know that the field contains a date value and the format in which that value is stored. There are many formats that you can use and are used in different databases; here are the available formats (case sensitive):

- |            |           |            |
|------------|-----------|------------|
| ■ yyMMdd   | ■ MMddyy  | ■ yyyyMMdd |
| ■ MMddyyyy | ■ cyyMMdd | ■ cyyMM    |
| ■ cyyDDD   |           |            |
|            | ■         |            |

All of these formats are used as shown above and allow the application to handle any date input. Additional examples:

- |                       |                       |
|-----------------------|-----------------------|
| ■ Field Class Setting | ■ Example Date Format |
|-----------------------|-----------------------|



- planetj.database.field.DateField, 2005/03/22  
yyyyMMdd
- planetj.database.field.DateField, 01/13/2005  
MMddyyyy
- planetj.database.field.DateField, 23/01/2005  
ddMMyyyy

■

If you wish to have the value of a DateField set to the current date, set the Default Value to \*CURRENT.

### Special Case: Multi-field Dates

A commonly encountered date scenario for customers is to have date information stored in multiple database fields. For example, individual fields that store century (19, 20), year (95, 06), month (1-12), and day (1-31). Currently, this cannot be handled through the Field Descriptors, so we'll use the operation's SQL to remedy the situation.

Often, during SQL selection, it is desirable to allow the end user to select the multi-field date as a single date field. Doing so can also enable the use of SQL data arithmetic which is very powerful. In the example below, we will use a multi-field date in the WHERE clause of an SQL operation. To enable this date selection, we need to build a String in a format such as "2004-4-3" and use the DATE function on it. First, add the following SQL segments to your test operation: `SELECT...WHERE...DATE ((CENTURY * 100 + YEAR) || '-' || MONTH || '-' DAY) BETWEEN ?555 AND ?555` Since you are concatenating fields to produce a date, WOW doesn't know what type of prompt you need. Therefore, you must create a derived Field Descriptor and set its data type to DATE. Substitute the ID of the derived FD in place of 555 in the SQL above. WOW will then know to prompt with a date picker. Doing this will also allow some very nice "fixed date" queries such as:

- **Email** - In the database an email column's value might be support@planetjavainc.com. By setting the Field Class to Email, the proper display value will be generated automatically. Also, the proper validation will automatically be performed to check for correct email address syntax. In particular, it ensures the value contains an '@' symbol as well as a '.' in the domain. It will also create the HTML mailto link automatically for that display value: `<a href="mailto:support@planetjavainc.com">support@planetjavainc.com</a>` Thus, a user could click on the link, then type and send an email to the address. This would especially be beneficial if you wanted to display a list of emails. Rather than manually coding each email link, by setting the Field class to Email, they would automatically be generated.
- **Fax Number** - The validation of a fax number field ensures that the length of the fax number is the correct length and only contains digits. If the value contains (, ), or - they are also accounted for.
- **First Name** - The validation of a first name field ensures that the field is not blank, has at least 2 characters, and is not completely numeric.
- **Gender M/F** - If no other Possible Value Key or Operation is specified, this Field Class uses the Possible Value Key \*GENDER\* to return a list of genders (Male, Female).

- **HTML Code** - Allows the field value to contain HTML tags and be rendered as HTML code by the browser. Otherwise, those tags would be mostly likely stripped out, leaving only the text.
- **Image URL Reference** - This class takes the field's value and uses it as the source for an HTML image tag. For instance, if the field contained the value "images/planetj.gif", it would be sent to the browser as the source attribute of an image tag: ``. This allows the user to visually display the contents of a field containing image references.
- **Last Name** - The validation of a first name field ensures that the field is not blank, has at least 2 characters, and is not completely numeric.
- **Password** - This field class ensures that when a password field is displayed, it will be replaced by asterisks. This can be very useful for sensitive information such as passwords.

#### Field Class Parameters

If you manually enter the Field Class, you can optionally enter special parameters. These parameters indicate features or attributes of the Field Class. The following are Field Class parameters that you can enter for a PasswordField only: `digitRequired=true` - This forces the user to enter at least one digit/number in the passwordExample:

```
planetj.database.field.PasswordField,minLength=7,digitRequired=true
```

In the example above, a valid password would be: abcd123. Invalid passwords would be: abc123 (too short), abcdefgh (no digits).

- **Phone Number** - The validation of a phone number field ensures that the length of the phone number is the correct length and only contains digits. If the value contains (, ), or - they are also accounted for.
- **Social Security Number** - The validation of a social security field ensures that the length of the social security number is correct and its value contains only digits. If the value contains any -, they are also accounted for.
- **State** - If no other Possible Value Key or Operation is specified, this Field Class uses the Possible Value Key `*US_STATES*` to return a list of all US states.
- **T/F Boolean** - This field can be used when you want the value in the database to be of type CHAR and length 1. Since the field is a boolean field, it will be displayed in the form of a checkbox. Thus, the user cannot enter the wrong value. When setting the value programmatically, a boolean true or false can be used, which will in turn set the value to T or F.
- **Timestamp** - Stores date and time values in ISO format (yyyy-MM-dd-HH.mm.ss.nnnnnnnnn) and displays to the user in a more readable format (MM/dd/yyyy HH:mm:ss). Use this field in conjunction with `*CURRENT*` as the field's Default or Auto Update Value to automatically insert the current date and time.
- **Upper Case** - This field ensures that its value is always upper case both when displaying and inserting or updating to the database.
- **URL Reference** - For fields containing URL references (e.g. `http://www.google.com`), this field class wraps the field's value in HTML anchor tags so that it is rendered to the user as a hyperlink rather than plain text.

- **User ID** - Designates the field as a user ID, which is needed when the field is used for authentication or authorization. No specific validation is performed.
- **YBlank Boolean** - Same as T/F Boolean, except uses 'Y' and ' ' instead of 'T' and 'F'.
- **YN Boolean** - Same as T/F Boolean, except uses 'Y' and 'N' instead of 'T' and 'F'.
- **Zip Code** - The validation of a zip code ensures the zip code is correct length and contains only digits. If the value contains a -, it is also accounted for.
- **Zip Code Suffix** - The validation of a zip code suffix field ensures the zip code suffix is the correct length and contains only digits. If the value contains a -, it is also accounted for.
- **Field Descriptor Type** - This is where you can select which field descriptor type, if any, you would like to be using.
  - Default - This descriptor type is the default setting and applies to most FD definitions
  - Derived - The FD represents a derived field used in SQL statements (e.g. `View as d_viewfld` where the text View is shown for each row and the field is represented with the name `d_viewfld`). A derived field does not exist in the table and its value is used for display only.
  - Table Descriptor - This field descriptor represents a table (instead of a field) and has the same name as the table, except it is prefixed with a tilde (~). There can only be one table descriptor for each table. The table descriptor is created automatically by WOW when "Create FDs" is selected.
- **Formatter Class** - Specify the Java class that will be used to format the data for a report. This setting only affects how the data is displayed. For example, setting a CHAR 10 to a "Phone Number" would result in 1234567890 to be shown as (123)456-7890. Some examples of formatter classes include:
  - ISO Date - This formatter class will change the format of the field to ISO YYYY-MM-DD format. To set up this formatter, enter `planetj.formatters.DateFormatterISOYearMonthDay` in the Field Descriptors Formatter Class field.
  - Day/Month/Year Date - This formatter class will change the format of the field to DD/MM/YYYY format, the format generally used throughout Europe. To set up this formatter, enter `planetj.formatters.DateFormatterDayMonthYear` in the FD formatter class field.
  - Day.Month.Year Date - This formatter class will change the format of the field to DD.MM.YYYY format, the format generally used throughout Europe with the EURO separator as well. To set up this formatter, enter `planetj.formatters.DateFormatterEUDayMonthYear` in the FD formatter class field.
  - Locale-specific Date - This formatter class will change the format of the specified field to the locale-based standard date format, utilizing the locale set on Apache Tomcat's Java settings. To set up this formatter, enter `planetj.formatters.LocaleSpecificDateFormatter` in the FD formatter class field.
  - German Number - This formatter class will change the format of number fields to use a period for the separator and a comma as the decimal, as is done in Germany and some other places in Europe. To set up this formatter, enter `planetj.formatters.GermanNumberFormatter` in the FD formatter class field.
  - Locale-specific Number - This formatter class will change the format of the specified field to the locale-based standard number format, utilizing the locale set on Apache Tomcat's Java settings. To set up this formatter, enter

planetj.formatters.LocaleSpecificNumberFormatter in the FD formatter class field.

- **Concurrency** - [EE] Regulates the concurrent updating or deleting of a row or field by users. A concurrent update occurs when user A views a field, user B then updates that field, and then user A tries to update that same field, overwriting user B's changes. A concurrent delete is when user A views a field, user B updates that field, then user A deletes the row, erasing user B's changes. You can adjust this setting to allow or disallow concurrent updates and deletes on the described field.
  - NO\_CONCURRENT\_ALTERATIONS\_ALLOWED - If user A reads a field with this concurrency value from the database, and then user B makes changes to the field, user A cannot update the field or delete the row containing the field without first rereading it.
  - CONCURRENT\_DELETES\_ALLOWED - If user A reads a field with this concurrency value from the database, and then user B makes changes to the field, user A is allowed to delete the row containing the field without first rereading it.
  - CONCURRENT\_UPDATES\_ALLOWED - If user A reads a field with this concurrency value from the database, and then user B makes changes to the field, user A is allowed to update the field without first rereading it.
  - CONCURRENT\_UPDATES\_AND\_DELETES\_ALLOWED - If user A reads a field with this concurrency value from the database, and then user B makes changes to the field, user A is allowed to update the field or delete the row containing the field without first rereading it. This is the default value.
- **Getter Method** - [EE] This attribute is only used for Derived Fields. The DataEngine parses the getter method String to get the method to call and parameters to use. Then Java reflection is used to invoke the proper method to get the Derived Field's value.
- **Setter Method** - [EE] This attribute is only used for Derived Fields. The DataEngine parses the setter method String to get the method to call and parameters to use. Then java reflection is used to invoke the proper method to set the correct Field(s) value.
- **Association Operation** - The pull down for this attribute will show any associated operations (1-1, 1-many, etc.) available. Setting this attribute to an operation causes that associated operation to execute when the field is clicked on.
- **Notify Status Change** - Tells whether or not changes the user makes to this field value will triggers a "status change" or not. A status change notifies the server that the field has been given a new value, and allows the server to re-render either the entire page, or other fields on the page.
  - **No** - Do not notify the server when the field's value changes. This is the default value
  - **Yes** - Notify the server whenever the value of the field changes. The round-trip back to the server will re-render the entire page, and any possible value operations shown on the page will be executed again. Therefore, if another field's possible values operation depends on the value of this field, the possible value operation will be executed again, but this time it will contain the new value.
  - **Ajax** - Notify the server whenever the value of the field changes. Instead of re-rendering the entire page, only the fields in the current row will be re-rendered. Any new values or possible values for those fields will be sent back to the browser from the server, and those fields on the screen will be updated without reloading the whole page.
- **Remarks** - User documentation for field only.
- **XML Tag** - This defines the XML tag to be used for this field when the XML icon is

clicked.

## Authorization Settings

 **Authorization Settings**

Read Authorization Operation

-- None --



Edit Authorization Operation

-- None --



- **Read Authorization Operation** - [PRO] Used to limit the users who can read the data for this field.
- **Edit Authorization Operation** - [PRO] Used to limit the users who can edit the data for this field.

## Additional Settings

 **Additional Settings**

Sortable

☒

Auto Increment

WOW Driv

Read Only

☐

Auto Trim On Read

☒

Currency

☐

Auto Trim On Write

☐


Id

123154

Usage Id

☒ -- Non

☐

 **Database Settings**

System Alias

SAMPLES

Column Size

8

Library Name

PJDATA

Scale

0

Table Name


EMPLOYEE

Nullable

Allow Null

SQL Type

NUMERIC



Key Position

SQL Type Name

NUMERIC

- **Sortable** - Whether or not results can be sorted by the values in this field. In the results table, sortable fields have small up and down arrows next to their names; the user can click on these buttons to sort by that field.
- **Auto Increment** - For fields that need auto-incremented unique values.
  - None - No auto-increment.
  - Database Driven - The value for each row will increment automatically by the database system being used. This should be selected for database identity

fields.

- WOW Driven - The value for each row will be incremented automatically as it is inserted into the database. By selecting this option, you allow WOW to increment each row on its own, without any intervention.

The Default value under the Basic Settings can be used to set the starting value for this option.

- **Read Only** - Whether or not the field is read-only. Read-only fields cannot be updated.
- **AutoTrim On Read** - If spaces on the end of field values should be automatically trimmed and discarded as the values are read from the database.
- **AutoTrim on Write** - If spaces on the end of field values should be automatically trimmed and discarded as the values are written to the database.
- **Currency** - Whether or not this field represents a currency or monetary value. It's the same as setting the Formatter Class to "Currency". Checking this box causes the value to be right justified.
- **Id** - A unique number associated with this field descriptor. This is the ID that can be used when doing field descriptor prompting.
- **Usage Id** - An integer that is associated with this database column. This is useful when two columns in different tables have different names, but represent the same logical type of data. They can then be referred to by usage ID instead of by name.

## Database Settings

- **System Alias** - The name of the alias being used for the current connection.
- **Column Size** - The amount of information the described field can contain. This setting specifies the maximum number of characters or digits that can be stored in this field's database column.
- **Library Name** - The name of the library that the described field is located in. If the Field Descriptor is shared among tables in multiple libraries, this field will contain an asterisk.
- **Scale** - For a numeric type field, this is the number of digits to the right of the decimal point.
- **Table Name** - The name of the table that the described field is located in.
- **Nullable** - Whether or not the field can contain null values.
- **SQL Type** - This is the corresponding SQL data type for the field.
- **SQL Type Name** - This is the corresponding SQL data type for the field.
- **Key Position** - Defaults to 0 for non-key fields. This field represents the position of this field in the table's key. For instance, if this table (Table Name) has only one key, and this FD is the key field, then the key position should be 1. If this table's key is made up of two columns, you need to set the FDs appropriately. If a Field Descriptor is not a key, this value should be 0. Keys are used to identify a unique record (row) within the table. Only one row is allowed to exist with each key (or set of key values).

## Field Descriptor Views

When you are viewing field descriptors in Field Descriptor Manager, by default the ID, Field Name, External Name, Required, and Field Class columns are the only columns displayed (these columns are explained in greater detail at the beginning of this chapter). Although you can always choose to view the details of a Field Descriptor to see all of its settings, there are times when you will want to view different columns of multiple field descriptors. There are eight different views or filters that can be used within Field Descriptor Manager to control the columns that are displayed in the table of field descriptors. These views are found on the drop down menus of the Field Descriptor Manager and are described below in the next five sections.

### Table FD's

- **Auto Trim** - Displays the Auto Trim On Write and Auto Trim On Read columns for all the field descriptors in the current table and library.
- **Default Value** - Displays the Default Value field of all the field descriptors in the current table and library.
- **Display Properties** - Displays the Field Set, Display Order, Display Rule, and Display Components fields of all the field descriptors in the specified table and library.

### Shared FD's

- **Table FD's** - Shows all field descriptors for the current table that are shared. Shared field descriptors will not be shown for a table and library by simply clicking the View FD's button. This is because shared field descriptors do not correspond to any specific library. You must use the Table FD's option to view the field descriptors for a table that have been shared.
- **All FD's** - Displays the shared field descriptors for all tables for the current system alias.

### Usage IDs

- **Table FDs** - Shows the Usage ID field for all the field descriptors for the current table and library.
- **Table FDs w/UsageId** - Shows the Usage ID field for all the field descriptors for the current table and library which have a Usage ID.
- **All FDs w/UsageId** - Shows the Usage ID field for all the field descriptors for the current system alias which have a Usage ID.

### Search By

- **Multiple Fields** - This search option provides a means to display a subset of field descriptors based on one or more of the following search fields: Library Name, Table Name, Field Name, External Name, Id, Usage ID, Field Descriptor Type.

### Quick Edit

The quick edit feature lets you display a subset of field descriptors and edit specific fields from the table view. Each of the views below shows a different set of fields:

- **Default Value and Required** - Shows an editable table that includes the Default

Value and Required fields.

- **Display Properties** - Shows the following field descriptor fields: Field Set, Display Order, Display Rule, Display Component, Id, Field Name, and External Name.
- **Key Position** - Shows the following field descriptor fields: Key Position, Id, Field Name, External Name.



## Prompting Using Field Descriptors

When an SQL operation that has a question mark in its code (e.g. `SELECT * FROM PLANETJ.CUSTOMER WHERE BALANCE > ?`) is run, the Field Descriptor for the `BALANCE` field is used to generate an input where the user can provide a value for the query at runtime. If the Field Descriptor indicated that the maximum value for this field was 9,999,999 then the HTML input would not allow the user to enter a larger value. If the Field Descriptor indicated that the field should be displayed with an HTML text area component (instead of a text field), then the generated input would be a text area.

You can override this behavior and use a different Field Descriptor for generating the user prompt. To do so, append the ID of the Field Descriptor you want to use to the question mark. For example, the query `SELECT * FROM PLANETJ.CUSTOMER WHERE BALANCE > ? 307` would use Field Descriptor 307 for generating the prompt input shown to the user. This enables you to present a prompt for the same field in different way in different queries.

# WOW Features








## Derived Fields

Perhaps one of the most commonly used features in WOW, derived fields provide the ability to create a "virtual" field. This field is defined by a field descriptor, just a like a normal field, yet it does not actually exist in a physical file. It is simply a container field that can be used to return any value needed. It can be handled and manipulated just a like an actual database field.

A derived field is represented in SQL by a column name alias. For instance, the following SQL statement contains a derived field with the column name alias D\_DETAILS:

```
SELECT 'Details' AS D_DETAILS, FIRSTNME, LASTNAME FROM PJDATA.EMPLOYEE
```

This statement returns three columns, in the following order: D\_DETAILS, First Name, and Last Name.








	D_DETAILS 	▲ First Name ▼ 	▲ Last Name ▼ 
	Details	ERICA	PINEIRO
	Details	Laura	Klocke
	Details	Ted	Cessna
	Details	Paul	Thomas

Notice the first part of the column name alias ( 'Details' ) represents the initial value that is displayed in the derived field. A string, which requires single quotes, was used in this example but you could also instead use numbers, functions, other field names, etc. The second part ( AS D\_DETAILS ) assigns an alias (column name) to the value defined in the first half.

Another method of creating a derived field in WOW is to use the !!<field name> notation within the SQL statement similar to this:

```
SELECT !!D_DETAILS, FIRSTNME, LASTNAME FROM PJDATA.EMPLOYEE
```

This statement returns three columns, in the following order: D\_DETAILS, First Name, and Last Name.

	▲ Details ▼ 	▲ First Name ▼ 	▲ Last Name ▼ 
		ERICA	PINEIRO
		Laura	Klocke
		Ted	Cessna
		Paul	Thomas

Notice the fields in the Details column are all blank, unlike the first example which required you to set a value in the field or pull a value from another field in the database.

This method allows us to include a derived field in a result table without requiring you to pull a value from the database or set value on the field. This method is especially useful

when creating derived fields for Tabbed Operations. Please note that this method requires you to create a derived field descriptor before the operation will run properly.

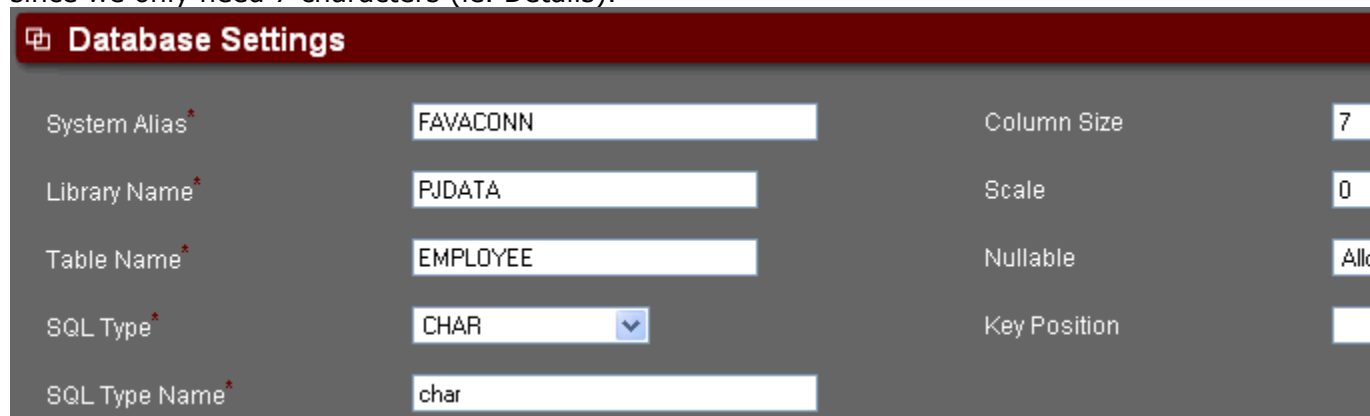
## Creating a Derived Field Descriptor

Once you have the derived field defined in your SQL, you can format and manipulate it by creating its derived field descriptor. This will allow you to assign it default values, associations, field classes, and so on. A derived field descriptor differs from a normal field descriptor in really only one way: its Field Descriptor Type is set to 'Derived'. This allows WOW to handle it appropriately. Also, a field descriptor for a derived field will not automatically generate by clicking the 'Create FDs' button in the Field Descriptor Manager. Thus, you must manually create derived field descriptors.

The easiest way to do this is to simply copy another field that is of the desired data type. For instance, if your derived field was a calculation that returned a decimal number, you would copy an existing field descriptor for a field of type DECIMAL or NUMERIC. To continue our example from the previous section, here are the steps to creating a derived field descriptor for D\_DETAILS:

1. **Create a Field Descriptor**

Since the only value that D\_DETAILS will hold is the string 'Details', a good field descriptor to copy (rather than creating one from scratch) would be a simple CHAR field. Now, navigate to the Database Settings section of this new field descriptor and change the values to reflect those shown in the figure below. In particular, the Library and Table Names are the same as specified in the SQL statement listed above. Also, SQL Type and SQL Type Name specify the CHAR field type and the Column Size is 7 since we only need 7 characters (ie. Details).





The screenshot shows a 'Database Settings' form with the following fields and values:

Field	Value	Field	Value
System Alias*	FAVACONN	Column Size	7
Library Name*	PJDATA	Scale	0
Table Name*	EMPLOYEE	Nullable	All
SQL Type*	CHAR	Key Position	
SQL Type Name*	char		


2. **Set the Field Descriptor's Settings**




Besides the Database Settings given in Step 1, there are two other Field Descriptor settings that are key to success. The first is the Field Name field under the Basic Settings section. In this example, the Field name is D\_DETAILS.

 **Basic Settings**

Field Name*	<input type="text" value="D_DETAILS"/>	External Name	<input type="text" value="Details"/>
Required	<input type="checkbox"/>	Required On Search	<input type="checkbox"/>
Default Value	<input checked="" type="radio"/> -- None --  <input type="radio"/> <input type="text"/>	Auto Update Value	<input type="checkbox"/>








The second is the Field Descriptor Type under the Advanced Settings section. This must be set to 'Derived'.

 **Advanced Settings**

Field Class	<input checked="" type="radio"/> -- None --  <input type="radio"/> <input type="text"/>	Formatter Class	<input checked="" type="radio"/> <input type="text"/> <input type="radio"/> <input type="text"/>
Field Descriptor Type*	<input type="text" value="Derived"/> 	Concurrency*	<input type="text" value="Concurrent"/>
Getter Method	<input type="text"/>	Setter Method	<input type="text"/>
Association Operation	<input type="text" value="-- None --"/> 	Notify Status Change	<input type="text" value="No"/>

### 3. Update and Run Application

Click Update and run the application. If everything has been done correctly, the column label will say 'Details' rather than 'D\_DETAILS'. This means the derived field has successfully been linked with the derived field descriptor and is picking up the External Name.

	▲ Details ▼ 	▲ First Name ▼ 	▲ Last Name ▼ 
	Details	ERICA	PINEIRO
	Details	Laura	Klocke
	Details	Ted	Cessna
	Details	Paul	Thomas

## Parameters

There are many cases where the statement you want to run cannot be completely specified at design time. This usually happens when the statement contains certain values that either needs to be directly inputted by the user at runtime or depend on the context in which the statement is being run (the context includes things such as the user's sign on information and previous statements that the user has run). WOW handles these cases by using default parameters. A parameter is represented in code by one or more question marks, possibly followed by additional parameter control characters. For example, the following SQL statement contains 3 different parameters:

```
SELECT * FROM PLANETJ.CUSTOMER WHERE (BALANCE > ? AND ID = ???CUSNUM) OR ??1 < 0
```

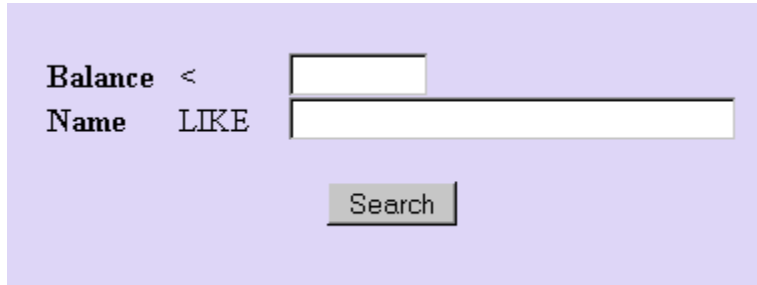
The `?`, `???CUSNUM`, and `??1` all serve as placeholders for values that are not known at design time, but will be plugged in to the statement at runtime before it is executed. This section will describe the various parameter types that are available in WOW and how to use them.

### SQL Prompt Parameters

A single question mark in an SQL statement represents a SQL prompt parameter. When a statement containing one or more SQL prompt parameters is executed, the user is prompted to enter values for these parameters. For example, when the statement:

```
SELECT NAME, BALANCE FROM PLANETJ.CUSTOMER WHERE BALANCE < ? AND NAME LIKE ?
```

is run, the user is shown the following screen.



The screenshot shows a user interface for entering SQL prompt parameters. It has a light purple background. On the left, the text 'Balance <' is displayed above 'Name LIKE'. To the right of 'Balance <' is a small rectangular input field. To the right of 'Name LIKE' is a longer rectangular input field. Below these two input fields is a rectangular button with the word 'Search' written on it.

After supplying values, the user can click the search button to run the statement with the values that were entered. Unlike most types of parameters which can be used in any type of operation, SQL prompt parameters can only be used in SQL operations.

### Field Descriptor Prompt Parameters

A field descriptor prompt parameter is similar to an SQL prompt parameter in that it is used to display an entry field for the user to supply a value for the parameter. The difference between the two is how WOW generates the entry field. For SQL prompt parameters, WOW determines which field descriptor to use for the entry field based on the SQL statement. For a field descriptor prompt parameter, WOW will use a specific field descriptor you specify to generate the entry field.

Field descriptor prompt parameters are denoted by a single question mark followed by the ID of the field descriptor to use. For example, the SQL statement:

```
SELECT * FROM PLANETJ.CUSTOMER WHERE BALANCE > ?49
```

will use the field descriptor with an ID of 49 to generate the prompt shown to the user. Field descriptor prompt parameters can only be used in SQL statements.

## Row Parameters

A row parameter takes information from a row of data and plugs it into a statement. A row parameter is indicated by two question marks followed by a database column name. For example, if a database record describing a single employee has been selected from the EMPLOYEES table, and now information about that employee's department needs to be selected from the DEPARTMENT table, the SQL statement might look something like this:

```
SELECT * FROM PJDATA.DEPARTMENT WHERE ID = ??DEPT_ID
```

This statement assumes that ID is the key in the DEPARTMENT table, and that the "current row" (from the EMPLOYEE table) contains a column called DEPT\_ID which is a foreign key to the DEPARTMENT table. When this statement is run, the value of the DEPT\_ID field of the "current" row is used as the parameter value.

**NOTE:** This parameter is automatically filled in by WOW; the user is not shown any type of prompt.

## User Parameters

A user parameter is similar to a row parameter, except instead of taking information from the "current" row, the information is taken from a row of data associated with the current application user. A user parameter is identified by three question marks in a row followed by a database column name. So, the following statement:

```
SELECT * FROM PLANETJ.CUSTOMER WHERE ID = ???CLIENT_ID
```

will select rows where the ID field is equal to the CLIENT\_ID field associated with the current user.

There is a special user parameter called USERID which is always associated with the id that was used to sign onto the application. This user parameter can be used with any type of application sign-on (except for an unsecured sign-on, which does not require the user to enter a user id or password). The SQL statement:

```
SELECT * FROM PLANETJ.USER_INFO WHERE ID = ???USERID
```

would select every row from the USER\_INFO table where the ID column has a value equal to the user ID of the current user. User ID's are always converted to uppercase, so in the above example all values in the ID column should be uppercase as well.

## Usage ID Parameters

In order to use a row or user parameter, you have to know the database column name of the field whose value you are interested in. In some cases this is not possible - usually this happens when multiple tables contain the same logical piece of information in different fields. In this situation, you can identify the field to use by its usage ID instead of its column name. A usage ID is an integer you can associate with one or more field descriptors. A usage ID parameter will look for a field descriptor with the specified usage ID in the row (either the user row or the current row) and use the value in the field described by that field descriptor.

A user usage ID parameter is denoted by three question marks followed by a caret and the usage ID. The statement:

```
SELECT * FROM PLANETJ.CUSTOMER WHERE ID = ???^18
```

would take the value associated with usage ID 18 in the user row as the parameter value. A row usage ID parameter is denoted by two question marks followed by a caret and the usage ID.

```
SELECT * FROM PLANETJ.CUSTOMER WHERE ID = ??^46
```

would take the value associated with usage ID 46 in the current row as the parameter value.

## Table Parameters

A table parameter is used when you want to allow the user to specify the table or tables to run an SQL statement against. For example, you might have multiple tables containing customer orders - every table would have the same structure but be specific to a single customer. You could then build a query which could apply to any of the tables - the user will pick the exact table to run the query against at runtime:

```
SELECT * FROM ?~PLANETJ.CUSTOMER WHERE ORDER_NUMBER = ?
```

A table parameter begins with a question mark followed a tilde (~) and includes the name of a table; in the above statement ?~PLANETJ.CUSTOMER is the table parameter. At runtime, the statement will be executed against whatever table the user specifies, which may or may not be the PLANETJ.CUSTOMER table. However, the PLANETJ.CUSTOMER table will be used to identify the field descriptors which will be used to display the parameter prompts to the user. The prompt for the table parameter will be based off of the table descriptor for PLANETJ.CUSTOMER - this table descriptor can be used to specify a display name and a list of possible table values for the user to choose from. The prompt for the second parameter will be based off of the ORDER\_NUMBER field descriptor in the PLANETJ.CUSTOMER table - even if this is not the table the user specifies for the actual statement execution.

## Parameter Parameters

A Parameter parameter is a parameter which gets its value from another parameter in the same statement. Parameter parameters are used when multiple parameters in a statement must all have the same value. For example if you wanted to look up customer balances that are within \$200 of a certain value, your query might look like this:

```
SELECT * FROM PLANETJ.CUSTOMER WHERE BALANCE +200 > ? AND BALANCE - 200 < ??1
```

The first question mark is a normal SQL prompt parameter - the user will be prompted for this value. The second pair of question marks is immediately followed by a number, indicating that it is a Parameter parameter. The user will not be prompted to supply a value for this parameter. Instead it will have the exact same value as the first parameter in the statement.

In general, a Parameter parameter is denoted by two question marks followed by a number. The number indicates which parameter in the statement should be used to supply the value (in the above example, the number 1 indicates that the first parameter should be used to supply the value).

## Context Parameter Parameters

A Context Parameter parameter is a parameter that is similar to a Parameter parameter,

but rather than getting its value from another parameter in the same statement, it gets its value from a parameter (search) in an associated statement. When Context Parameter parameters are used, parameters in an association need to have the same value as the parameters in the original SQL. For example, an original query might show a summary of a customer's balance between a certain date range. The query would also contain an association (using a derived field descriptor) that gets transaction details for that customer. The association (2nd SQL listed below) would thus need to use the same search date range:

```
SELECT CUSTOMER_NAME, SUM(AMOUNT), !!DETAILS FROM  
PLANETJ.CUSTOMER_TRANSACTIONS WHERE TRANSACTION_DATE BETWEEN ? AND ?
```

```
SELECT TRANSACTION_ID, AMOUNT FROM PLANETJ.CUSTOMER_TRANSACTIONS WHERE  
CUSTOMER_NAME = ??CUSTOMER_NAME AND TRANSACTION_DATE BETWEEN ??&1 AND ??&2
```

In the association, the first parameter is a Row parameter used to ensure the proper customer information is retrieved. The last two parameters are the context parameter parameters used to get the same date range to search on as the original query.

In general, a Context Parameter parameter is denoted by two question marks followed by an ampersand ('&') and a number. The number indicates which parameter in the original statement should be used to supply the value.

### **Using Context Parameter parameters in Possible Values**

Another use for Context Parameter parameters is to reference search parameter values within possible value operations assigned to search parameters. An example scenario:

You have an operation with 2 search parameters that shows a report with a list of employees. The 1st parameter is Company divisions and the 2nd is company departments. When a user selects 1 or more divisions, you want the possible values for departments to only reflect the chosen divisions. In a 2nd example below, employees can be searched on by work department and job. As you change your work department selection (1 or more), the list of Jobs to choose from changes:

The main operation with the 2 search fields looks like this:

```
SELECT * FROM PJDATA.EMPLOYEE WHERE WORKDEPT in ? and JOB in ?
```

Next, possible values operation are created for the fields WORKDEPT and JOB:

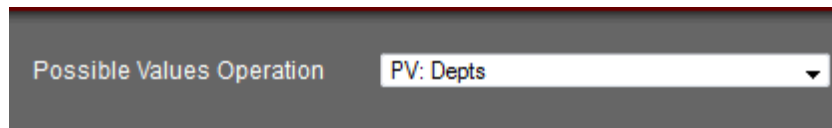
#### **WORKDEPT:**

#### **JOB:**

Notice the use of the context parameter ??&1 in the 2nd possible values operation (JOB). This tells WOW to substitute in the values of the 1<sup>st</sup> search parameter (work departments) into the possible values statement above (JOB).

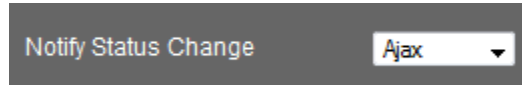


For field WORKDEPT, the Field Descriptor has the 1<sup>st</sup> possible values operation assigned:



Possible Values Operation PV: Depts ▼

And Notify Status changed to Yes or Ajax:



Notify Status Change Ajax ▼

The notify status change setting (Ajax or Yes) tells WOW to rerun the possible values operations assigned to the 2 search parameters whenever the 1<sup>st</sup> search parameter has it's selection changed.

For field JOB, the Field Descriptor has the 2<sup>nd</sup> possible values operation assigned:



Possible Values Operation PV: Jobs by Depts ▼

## Runtime Parameters

Runtime parameters are parameters which are specified when the user first enters an application, and can then apply to all operations executed by that user. For example, let's say you have sales offices in three different locations: Atlanta, Boston, and Cleveland. You want to develop a WOW application containing various operations which let people from each branch run different queries against sales made by their branch. You could include the branch name in each query using regular SQL parameters like this:

```
SELECT * FROM PJDATA.SALES WHERE BRANCH = ? AND AMOUNT > ?
```

```
SELECT * FROM PJDATA.SALES WHERE BRANCH = ? AND DATE = ?
```

```
SELECT * FROM PJDATA.SALES WHERE BRANCH = ? AND ACCOUNT = ?
```

The only problem with this scenario is it forces users to select their branch for every query that is run. If you rework these queries to use runtime parameters instead, then the branch can be specified once when the application starts up and used for all subsequent queries without further user input.

Two question marks followed by a colon ":" and an identifying name is the sequence used to indicate a runtime parameter. Using runtime parameters for the branches in the above queries gives:

```
SELECT * FROM PJDATA.SALES WHERE BRANCH = ??:BCH AND AMOUNT > ?
```

```
SELECT * FROM PJDATA.SALES WHERE BRANCH = ??:BCH AND DATE = ?
```

```
SELECT * FROM PJDATA.SALES WHERE BRANCH = ??:BCH AND ACCOUNT = ?
```

To specify a value for the BCH runtime parameter, the application should be started with a URL like this:

```
http://www.planetjavainc.com/wow/runApp?id=40&BCH=Atlanta
```

This starts up application 40 and indicates that "Atlanta" is the value for all runtime parameters named "BCH". The '?' denotes the start of parameters and '&' is used to separate parameters. Users from different branches can use links specifying their branch when starting the application:

```
http://www.planetjavainc.com/wow/runApp?id=40&BCH=Boston
```

```
http://www.planetjavainc.com/wow/runApp?id=40&BCH=Cleveland
```

When they run the operations, they will not have to select which branch they are querying.

## Request Parameters

Request parameters are parameters which get their values from the HttpRequest. For example, let's say you had some HTML (in a JSP or in Operation instructions) similar to this:

```
<input type="text" name="myInput" />
<input type="hidden" name="myHiddenInput" value="1998" />
```

In your operation you could then use a Request parameter to get the values from the HTML. Two question marks followed by a percent sign "%" and an identifying name is the sequence used to indicate a Request parameter. So, from the above example, if you used the parameter ??%myHiddenInput in your SQL, the value returned for the parameter will be 1998. If, in turn, you used ??%myInput as the parameter in your SQL, the value returned for the parameter would be whatever value the user entered in the input.

## Session Parameters

Session parameters are a more advanced parameter type and are only really used when doing custom programming. Two question marks followed by a semi-colon ";" and a key name is the sequence used to indicate a Session parameter. An example of a session parameter looks like this:

```
??;myParm
```

Using the above parameter as an example, WOW would look in the session for the key "myParm" and use the value stored in the session for that particular key as the value for the parameter.

## Special User Library Parameter

WOW supports the ability to store metadata in any number of user libraries or

schemas. The default is "PJUSERxx" where xx is the release id. You can specify the following to use the current user library.

```
SELECT * from PLANETJ_USR.SQLOPS
```

"PLANETJ\_USR" will be replaced by the actual default user library at runtime. "PLANETJ\_SYS" can be used for the current WOW System library.

NOTE: Changing WOW System data can result in a corrupt environment which is not covered by WOW Support agreements. It is recommended that this only be used for read only purposes.

## RowCollection Parameters [Minimum Version: WOW 7.0]

RowCollection parameters are parameters which get their values from the current RowCollection (1 or more Rows displayed by an operation). RowCollection parameters can be used to show attributes of the current rowcollection in your operation results, such as in the operation title or instructions. A RowCollection parameter typically is not for use in your SQL Statement, because the RowCollection is created after the SQL is run. Below are the supported RowCollection parameters:

- `?!WOW_RC_SIZE` - Substitutes in the number of rows currently displayed on the page.
- `?!WOW_SQL` - Substitutes in the SQL used to generate the results currently displayed. Useful when debugging a problem with the operation.

For example, if an operation displays 10 rows of data and the title = "Data for `?!WOW_RC_SIZE` Employees", the title would look like:

Data for 10 Employees

## Defaulting Parameter Values

Normally when user needs to fill in a parameter's value, that parameter will default to a blank value. For example, if your query is:

```
SELECT * FROM PJDATA.CUSTOMER WHERE BALANCE > ?
```

The prompt shown to the user would look something like this:



However, if you want your parameter to have a default value of 1000, you can specify this in your SQL statement. Using the code:

```
SELECT * FROM PJDATA.CUSTOMER WHERE BALANCE > ?{1000}
```

tells WOW that it should use a default value of 1000 for the parameter. Running an operation with the above code results in this prompt (before the user enters any data in):

A screenshot of a SQL prompt interface. It features a light blue background. On the left, the text "BALANCE >" is displayed. To its right is a white text input field containing the number "1000". Below the input field is a small, rectangular button with the word "Search" in a dark font.

The user can type in any value he or she wants; 1000 is just a default value. If your field has possible values and you want to use a default value, remember that you need to use the value you want as the default, not the display value.

In general, any parameter that is displayed to the user can be given a default value by appending the default value, enclosed in curly braces, onto the end of the parameter (there should not be any spaces between the rest of the parameter and the opening curly brace). Here are several more examples of SQL statements which assign default values to parameters:

```
CALL PLANETJ.MY_SPROC (?45, ?92{Red}, ?14{Orange})
```

```
DELETE FROM PLANETJ.EMPLOYEE WHERE LASTNAME = ?{Stewart} AND FIRSTNAME = ?
```

```
SELECT * FROM ?~PLANETJ.MYTABLE{PLANETJ.THISTABLE}
```

```
SELECT * FROM PLANETJ.EMPLOYEE WHERE HIREDATE BETWEEN ?{*current - 365 days}
and ?{*current}
```

**NOTE: '\*current' is a special value used by WOW to specify the current date. Manipulation can be done using 'days' as in the example above.**

## Operation Property Groups

The Properties field allows you to configure your operations in various ways. The screen shot below shows where the Properties field is located in the "Creating Operations" screen:

Display			
Allow Details	<input checked="" type="checkbox"/>	Display Group	Default
Allow Inserts	<input checked="" type="checkbox"/>	Display Order	
Allow Updates	<input checked="" type="checkbox"/>	Display Columns	
Allow Deletes	<input type="checkbox"/>		
Properties	<pre>DisplayColumns{ results;; details;; } DetailDisplay{addButtonsURI;; button locations;; buttonJustify;; colCnt;; colonAfterLabel;; copyURI;; delete;; deleteText;; editButtonsURI;; editURI;; grid;; insert;; insertAndCopy;; insertAndNew;; insertText;; insertURI;; label justify;; maxInputWidth;; maxInputWidthSum;; nextAndPrevious;; nextText;; previousText;; printURI;; tableWidth;; updateAndNextPrevious;; updateText;; viewButtonsURI;; viewURI;; } TableDisplay {selectionType:single; refresh:true; chart:true; excel:true; msWord:true; xml:true; editFD:true; print:true; sorting:true; drawGrid:true; rowCopy:true; updateable:false; deleteAll:false; nextPrevious:true;}</pre>		

Within the properties field are different property groups. These groups are used to change the look and feel of the tables and data selected by the SQL statement. For instance, a few of the different property groups available are DisplayColumns, DetailDisplay, and TableDisplay. The properties of each group will be listed between curly brackets {}. For each property, the name of the property is followed by a colon and then the value (or comma separated list of values), and finally by a semicolon. Below are samples of properties groups correctly formatted:

```
DisplayColumns{ results:*; details:*; }
```

```
DetailDisplay{colCnt;; copyURI;; editURI;; insertURI;; maxInputWidth;;
maxInputWidthSum;; printURI;;viewURI;;}
```

```
TableDisplay{ selectionType:none; refresh:true; chart:true; excel:false;
msWord:true; pdf:true; xml:true; editFD:false; print:true; sorting:true;
drawGrid:true; rowCopy:false; updateable:true; deleteAll:true;
nextPrevious:true; }
```

```
OperationLabels{searchDisplay:3;}
```

Note that whitespaces (new lines, spaces, and tabs) are irrelevant to property group formatting.

## AutoRun {}

[EE] This property group allows you to set the run schedule for an "Auto-Run – Batch Process" (an operation that is scheduled to run automatically when an application is

started).		
Property	Value	Description
startDate	TODAY   <i>MM/dd/yyyy</i>	The date the batch auto run operation should start.
startTime	IMMEDIATELY   <i>hh:mm</i>	The time the batch auto run operation should start.
frequency	<i>integer</i>	How often the batch auto run operation should execute in seconds (e.g. 900 = 15 minutes, 86400 = 1 day).

In the example below, the first run of the operation would occur on January 1, 2007 at 1 A.M., repeating every 7 days thereafter.

```
AutoRun {
  startDate:1/1/2007
  startTime:1:00 am;
  frequency:604800;
}
```

## Browser {}

This property group allows you to control the browser behavior when the operation is run. One use for these properties would be when an operation displays a small pop-up window.

Property	Value	Description
url	<i>URL</i>	URL to load in the window.
target	<i>window name</i>   _BLANK   _SELF   _PARENT   _TOP	Target where to load the window. The value "_SELF" would ensure that the operation runs in the same window. _BLANK will open a browser in a new window or tab.
width	<i>integer</i>	Width for the browser window.
height	<i>integer</i>	Height for the browser window.
toolbar	TRUE   FALSE	Show the toolbar.
location	TRUE   FALSE	Show the location bar.
menubar	TRUE   FALSE	Show the menu bar.
directories	TRUE   FALSE	Show the directories (links/bookmarks).
scrollbars	TRUE   FALSE	Show the scroll bars.
resizable	TRUE   FALSE	Allow resizing of the browser window.
copyhistory	TRUE   FALSE	Copy the browser history.

## Chart {}

[Deprecated. Chart is no longer support. See WOW Fusion Charts] This property group allows you to specify properties used to create and generate a chart using JFreeChart.

## Config {}

This property group allows you to specify a replacement library list for an operation.

## CSV {}

Specifies formatting information for CSV documents (CSV documents include Microsoft Word and Microsoft Excel formats). When a user chooses to view data in a CSV document, this property group describes how that document should be formatted.

Property	Value	Description
columnHeadings	INTERNAL   EXTERNAL	Indicates whether the internal database names or the external "user-friendly" names should be used for column names in the CSV. The default is INTERNAL.  <b>Example:</b> CSV { columnHeadings: external; }
outputRows	ALL   SCREEN   SELECTED	This would cause the Excel download to use external labels for column headings. Indicates which rows should be exported. Possible values are ALL (all rows which satisfy the query), SCREEN (only rows on the current screen), and SELECTED (only rows which the user has selected). By default, all rows are exported.
displayColumns	<i>field,field,...</i>	Indicates which columns should be exported. You may type in a comma separated list of column names that should be exported. By default, all columns returned by the query are exported.

## DetailDisplay {}

When a single entry (row) is displayed, this section contains information about how to display the details of a row to the user (row details are what the user sees when they insert a new row into the results, or when they select a row from the list of rows in the results, and choose to view, edit, or copy that row). Most SQL operations do not need to include these properties - they can simply use the WOW defaults. If you want to use a different JSP to display detailed results, set the appropriate DetailDisplay property. The features described below should only be used by advanced programmers who have experience with Java and JSP programming.

Property	Value	Description
addButtonsURI	<i>file path</i>	JSP to use for buttons during an insert.

button locations	top   bottom	Designates where the buttons are located on details screen. The default is to show buttons on both the top and bottom.
buttonJustify	RIGHT   LEFT	The buttons for the Detail screen are displayed at the top and bottom of the detail and have value such as Insert, Update, and Cancel. The default value is right.
cancel cancelText	TRUE   False <i>text</i>	Allow cancel on details screen. Text to be used on the Cancel button. <b>NOTE: Requires WOW version 6.6.1 and above.</b>
colCnt	<i>integer</i>	Number of columns to display when showing Row details (default is 2).
colonAfterLabel	TRUE   FALSE	Append colon after labels on the details screen. The default is false.
copyTargetWindow	<i>window name</i>   _BLANK   _SELF   _PARENT   _TOP	Describes how to use a new window for copying a row. For more details, see the <b>editTargetWindow</b> property.
copyURI	<i>file path</i>	The JSP to use when displaying copied database rows.
delete deleteText	TRUE   FALSE <i>text</i>	Allow delete on details screen. Text to be used on the Delete button.
detailsTargetWindow	<i>window name</i>   _BLANK   _SELF   _PARENT   _TOP	Describes how to use a new window for viewing, editing, copying, or inserting a new row. For more information, see the <b>editTargetWindow</b> property.
editButtonURI		JSP to use for buttons during detail viewing.



editTargetWindow

*window name* | \_BLANK |  
\_SELF | \_PARENT | \_TOP

Information about the window to use when a row is edited. If this property is omitted, then the main browser window is used to edit a row's details. When this property is specified, a new window will be used to edit a row's details. The value of this property can either be a name for the new window or a list of detailed information about the new window. For example, if the property is specified like this:

```
editTargetWindow:  
claimEdit;
```

Then when a row retrieved by this operation is edited, the editing will be done in a new window entitled "claimEdit." In general, the exact name which the new window is given does not matter; however, if there is already a window with that same name open, then that window is used instead of opening a new one. If the special name "\_blank" is used, then a new window is always opened:

```
editTargetWindow: _blank;
```

Alternatively instead of just specifying a name, a whole list of information about the new window can be specified.

```
editTargetWindow:  
name=_blank, height=600,  
width=400, status=yes,  
menubar=no,  
scrollbars=no,  
resizable=no;
```

The above example would cause the new window for editing to have a height of 600 pixels, a width of 400 pixels, a status bar, no scrollbar or menu bar, and not be resizable. Only the name

editURI	<i>file path</i>	value is required - the rest are optional and can be omitted if you want to use the defaults. The JSP to use when editing database rows.
grid	TRUE   FALSE	Use grid to display details.
insert	TRUE   FALSE	Allow insert on details screen.
insertAndCopy	TRUE   FALSE	Show the insert and copy buttons.
insertAndNew	TRUE   FALSE	Show the insert and new buttons.
insertTargetWindow	<i>window name</i>   _BLANK   _SELF   _PARENT   _TOP	Describes how to use a new window for inserting a row. For more information see the <b>editTargetWindow</b> property.
insertText	<i>text</i>	Text to be used on the Insert button.
insertURI	<i>file path</i>	The JSP to use when inserting new database rows.
labelJustify	TOP   LEFT	Determines where the field's label is to be located, top (above) or to the left of the field's display. The default is to the left.
maxInputWidth	<i>integer</i>	The maximum input width allowed for table display (default is 36).
maxInputWidthSum	<i>integer</i>	The maximum sum of the input widths in the table display (default is 72).
nextAndPrevious	TRUE   FALSE	Show the next and previous buttons.
nextText	<i>text</i>	Text to be used on the Next button.
previousText	<i>text</i>	Text to be used on the Previous button.
printURI	<i>file path</i>	The JSP to be used when printing database rows.
tableWidth	<i>integer</i>	The width (in pixels) to be used to display the details.
updateAndNextPrevious	TRUE   FALSE	Show the update and next buttons.
updateText	<i>text</i>	Text to be used on the Update button.
viewButtonsURI	<i>file path</i>	JSP to use for buttons during detail viewing.
viewTargetWindow	<i>window name</i>   _BLANK   _SELF   _PARENT   _TOP	Describes how to use a new window for viewing a row. For more details, view the <b>editTargetWindow</b> property.
viewURI	<i>file path</i>	The JSP to use when viewing database rows.

## DisplayColumns {}

There are two properties in the DisplayColumns group, results and details. An asterisk can be used as a value to display all data. In all the properties of this group, the values 'ALL' and 'NONE' may be used.

Property	Value	Description
details	ALL   NONE   *   <i>field,field,...</i>	Used to specify what columns you want displayed in a details (single row) display. You can view details of any entry (row) by clicking on the corresponding View icon for the entry (row). Like the results property, the details property can also take specific column names. For example:

```
DisplayColumns {  
  results: *; details:  
  empno,firstname,lastname,  
  sex; }
```

Now if you were to view an entry instead of showing all of the fields, it would only display the employee number, first name, last name, and gender fields.

detailsExclude	ALL   NONE   *   <i>field,field,...</i>	Columns to exclude from the details view.
editableResults	ALL   NONE   *   <i>field,field,...</i>	Used to specify what columns should be editable in the results view. Same functionality as the resultsEditable property. this must be used in conjunction with the updateable property in the TableDisplay property group:

```
TableDisplay {  
  updateable: true; }
```

results

ALL | NONE | \* | *field,field,...*

Used to designate which columns (fields) are displayed in the table when a set of rows are displayed. To display only specific fields, simply delete the asterisk and replace it with column names or column number values you want to display. Each column name should be separated by commas. For example:

```
DisplayColumns { results:
empno,firstname,lastname;
details: * }
```

The example would only display the employee number, first name, and last name fields of the table. Syntax is very important. The property groups are case sensitive. Each property group must start with a capital letter on each word with no spacing between them. The field names are no case sensitive though. Another technique to displaying certain fields in the table is by using numeric values instead of row names:

```
DisplayColumns { results:
3,1,2; details: *; }
```

In the example above, the row names were simply replaced with their corresponding number. For example, the above DisplayColumns setting would display the third column, then the first column, and lastly the second column.

resultsEditable	ALL   NONE   *   <i>field,field,...</i>	Used to specify what columns should be editable in the results view. Same functionality as the editableResults property. This must be used in conjunction with the updateable property in the TableDisplay property group:
resultsExclude	ALL   NONE   *   <i>field,field,...</i>	Columns to exclude from the results.

```
TableDisplay {
  updateable: true; }
```

## Email {}

[EE] This property group allows you to specify properties that are used for e-mailing.

Property	Value	Description
from	<i>integer</i>	The From ID.
password	<i>text</i>	The SMTP/POP3 account password.
pop3	<i>integer</i>	The POP3 (incoming) mail server IP address to use.
to	<i>integer</i>	The To ID(s).
cc	<i>integer</i>	The CC ID(s).
bcc	<i>integer</i>	The BCC ID(s).
replyTo	<i>text</i>	The address replies should be sent to.
smtp	<i>integer</i>	The SMTP (outgoing) e-mail server IP address to use.
subject	<i>text</i>	The e-mail subject.
user	<i>text</i>	The SMTP/POP3 account user name.

## FieldSet{}

For information on the FieldSet property group, see [Laying out out details screen](#).

## Join {} [PRO]

This is an optional property group that can be used when the operation query contains a join. The properties are used to alter the default WOW behavior for handling a join.

Property	Value	Description
----------	-------	-------------

updateTables	<i>table, table, ...</i>	[EE] A list of one or more tables that are updateable when a WOW operation contains a join. When an update is performed, only tables in this list are affected.
deleteTables	<i>table, table, ...</i>	[EE] A list of one or more tables that are deleteable when a WOW operation contains a join. When a delete is performed, only tables in this list are affected.
insertTables	<i>table, table, ...</i>	[EE] a list of one or more tables that are insertable when a WOW operation contains a join. When an insert is performed, only tables in this list are affected.
updateExcludeTables	<i>table, table, ...</i>	a list of one or more tables that are not updateable when a WOW operation contains a join. When an update is performed, only tables not in this list are affected.
deleteExcludeTables	<i>table, table, ...</i>	[EE] a list of one or more tables that are not deleteable when a WOW operation contains a join. When a delete is performed, only tables not in this list are affected.
insertExcludeTables	<i>table, table, ...</i>	[EE] a list of one or more tables that are not insertable when a WOW operation contains a join. When an insert is performed, only tables not in this list are affected.
transactions	TRUE   FALSE	[EE] Should transactions be used when a joined row is inserted/updated/deleted. By default, WOW will not use transactions for joined inserts. If you want WOW to issue your joined insert as a single transaction (which is recommended if your database supports transactions), you must use the Join property group to specify this.

checkAssocs	TRUE   FALSE	Check whether or not associated joins should be checked for an operation. The default is false. <b>NOTE:</b> Using this property can have performance implications and is not advised unless absolutely necessary.
-------------	--------------	---

## LayoutDisplay {}

This property group allows you to override the layout display properties for this operation.

Property	Value	Description
toc width	<i>integer</i>	Width of left side navigation.
css	<i>file path</i>	CSS file.
company text	<i>text</i>	Company name.
heading text	<i>text</i>	Heading text on page.
sub heading text	<i>text</i>	Sub-heading text on page.
help uri	<i>file path</i>	Help URI link.
title	<i>text</i>	Title text.

## OperationLabels {}

This property group allows you to specify how to organize the search parameters and prompts.

Property	Value	Description
button	<i>text</i>	The search/update button label text.
buttonImg	<i>file path</i>	New search button image file.
secondaryInstructions	<i>text</i>	Instructions for second set of parameters.
dropDownItemDisplay	NULL   <i>text</i>	Controls the search drop down text. Can change to anything you want including "— Choose —" or "NULL" if you don't want any other drop-down values but the actual values (default is: — ALL —).
dropDownItemOrder	TOP   BOTTOM	Controls the search drop down item order.
dropDownItemValue	<i>text</i>	Value for the drop-down item specified by the <b>dropDownItemDisplay</b> property. The value that is passed to WOW when that option is selected. You cannot specify a value for <b>dropDownItemValue</b> unless you also specify a value for the <b>dropDownItemDisplay</b> property.

searchDisplay

*integer | field,field,...*

The order and/or rows to display search parameters in application. You specify the number of prompts to be shown on each row (using the order specified in the SQL statement), or you can put all the search parameter field names with "|" to specify a new row and "," to delineate each field name. If you specify field names and order, you must list all fields that you would like to show up in Application. Only used for horizontal parameters.

## Horizontal Parameters

An example of one way to use the OperationLabels property group is horizontal parameters. It allows you to specify the order and number of search prompts on each row. To setup horizontal parameters, you must first specify the parameters JSP with `/dataengine/jsp/horizontal_gen_params.jsp` as shown below.

Advanced			
Connection Alias	-- None --	Operation Class	
Row Count	50	Row Coll. Class	
Row Class		Parameters JSP	me/jsp/horizontal_gen_params.jsp

Here is a screenshot of horizontal parameters with 4 fields horizontally:

`OperationLabels{searchDisplay:4;}`.

**Horizontal Params**

Horizontal Search   Specified Search   Standard Search

In this search example the search parameters are displayed with 4 prompts on one line.  
This is specified in the OperationLabels Property Group with the the property searchDisplay set to 4.  
`OperationLabels{searchDisplay:4;}`

Employee # =  Work Dept =  Phone # =  Last Name =

Here is a screenshot of horizontal parameters with specified fields:

`OperationLabels{searchDisplay: workdept,lastname,empno|phoneno;}`.





In this search example the search parameters are displayed in the order and positioning specified by `row()` and `field()` delineators. The four search parameters are specified in the `OperationLabels` Property Group with the the property `searchDisplay`.

`OperationLabels{searchDisplay:workdept,lastname,empno|phoneno;}`

Work Dept =  Last Name =  Employee # =

Phone # =

## OperationSettings {}

This property group is used in creating a possible values selector.

## OptionalSignon {}

This property group allows you to override some of the default features for an optional signon.

Property	Value	Description
userLabel	text	The default is "User:".
passwordLabel	text	The default is "Password:".
title	text	The default is "Optional Signon".

## Paging {}

Paging refers to the process of returning a specific number of records per "page" screen. This property group allows you to control if and how the paging is presented. It can either be specified for an application or an individual authentication operation (if specified in both places, the properties in the operation will take precedence).

Property	Value	Description
enabled	TRUE   FALSE	When set to false, paging links are not displayed. This does not necessarily mean that there isn't a Next or Previous page. This just means that if there are links, they will not be shown.
justify	LEFT   RIGHT	Sets on which side of the page the paging links are aligned.
firstAndLast	TRUE   FALSE	Determines whether or not to display the First and Last page links.
pageNumbers	TRUE   FALSE	Page numbers allow the user to jump to a specific page in the results. This property determines whether or not to display these page number links.

pageCount	<i>integer</i>	Used in conjunction with <b>pageNumbers</b> set to true. Specifies the number of page numbers to show at one time. For example, lets say there are 6 pages. If the page count was set to 3 and you were currently on page 4, only pages 3, 4, and 5 would be displayed. The default is to show all of them. Any negative number means to show all page numbers.
useText	TRUE   FALSE	Paging also allows the ability to include descriptive text of what page the user is currently on. When set to true, the default text displayed would be something like the following: "Displaying rows 4 - 6 of 16". The <b>text</b> property can be used to change what text is being shown. There are a few placeholder properties that can be used in the text as well.
text	<i>text</i>	Used in conjunction with the false <b>useText</b> property to control what is displayed for text. For example, you could specify "Displaying page %page of %totalpages." which would show something like "Displaying page 10 of 23." Additional placeholders are listed below:  %firstrow - the number of the first row being displayed on the screen. %lastrow - the number of the last row being displayed on the screen. %totalrows - the total number of rows available. %page - the current page number being viewed. %totalpages - the total number of available pages.
nextAndPrevious	TRUE   FALSE	Determines whether or not to display the Next and Previous page links.

## ParameterOperators {}

This property group allows you to override default display behavior of an operation's search prompts.

Property	Value	Description
like	<i>text</i>	This property can be used to replace the "LIKE" text next to a search parameter that uses a "LIKE" statement in the SQL. Leave this property blank (e.g. <i>like</i> ;;) to get rid of the operator label altogether.
=	<i>text</i>	This property can be used to replace the "=" text next to a search parameter that uses an "=" statement in the SQL. Leave this property blank (e.g. <i>=</i> ;;) to get rid of the operator label altogether.
<	<i>text</i>	This property can be used to replace the "<" text next to a search parameter that uses a "<" statement in the SQL. Leave this property blank (e.g. <i>&lt;</i> ;;) to get rid of the operator label altogether.
>	<i>text</i>	This property can be used to replace the ">" text next to a search parameter that uses a ">" statement in the SQL. Leave this property blank (e.g. <i>&gt;</i> ;;) to get rid of the operator label altogether.

## PDF {}

[PRO] This property group allows you to override how a PDF file is displayed.

Property	Value	Description
bottomMargin	<i>integer</i>	Bottom margin.
evenColor	<i>#hexColorCode</i>	Even color.
evenReportColor	<i>#hexColorCode</i>	Even report color.
fontSize	<i>integer</i>	Font size.
headerColor	<i>#hexColorCode</i>	Header color.
headerFontSize	<i>integer</i>	Header font size.
landscape	TRUE   FALSE	Sets page layout to landscape mode.
leftMargin	<i>integer</i>	Left margin.
oddColor	<i>#hexColorCode</i>	Odd color.
oddReportColor	<i>#hexColorCode</i>	Odd report color.

relativeWidths	<i>integer</i>	Relative widths controls how wide your PDF columns will be. If you have 4 columns, you might pass in 1,2,1,4 to have your second column be twice as wide as the first and third columns and your fourth column is twice as wide as the second. This does not affect the width of the table, just the columns within the table. So passing in 0.5,1,0.5,2 would have the exact same affect.
repeatTableHeader	TRUE   FALSE	Repeat table header.
rightMargin	<i>integer</i>	Right margin.
showGrid	TRUE   FALSE	Show grid.
topMargin	<i>integer</i>	Top margin.

## PleaseWait {}

This property group allows you to set the JSP used by the please wait function.

Property	Value	Description
js	<i>file path</i>	JSP to use. If no file path is specified in the <b>jsp</b> property area then the default please wait JSP will be used.
message	string message	Message to display from please wait screen. The default is 'Please wait while your query is being processed...'

The please wait page is normally used on larger queries or operations that may take longer than a few seconds to execute. Instead of showing the user a blank screen you will show them a specified please wait page that informs them the action is occurring. In the example below, we are not specifying a please wait page URL so that it uses the WOW default.

Specifying a PleaseWait property group in an operation.

Display			
Allow Details	<input checked="" type="checkbox"/>	Display Rule	-- None -- ▾
Allow Inserts	<input checked="" type="checkbox"/>	Display Location	<input checked="" type="radio"/> -- None -- ▾ <input type="radio"/> _____
Allow Updates	<input checked="" type="checkbox"/>	Display Group	<input checked="" type="radio"/> Default ▾ <input type="radio"/> _____
Allow Deletes	<input checked="" type="checkbox"/>	Display Order	500
Display Columns	<input type="text"/>		
Properties	<pre>       subviewWidth;          apduSubviewHeaderViews; apduFooter;       viewButtonsURI;       viewURI;;     }     TableDisplay{       selectionType:none;   refresh:true;     chart:true;       excel:true;           msWord:true;      xml:true;       editFD:true;         print:true;       sorting:true;       drawGrid:true;       rowCopy:true;     updateable:false;       deleteAll:false;     nextPrevious:true;     }     PleaseWait(jsp:;)           </pre>		

Allow Details

### Display Rule

Allow Inserts

Display Location

Allow Updates

Display Group

Allow Deletes

Display Order

Display Columns

## Properties

Running All Employees operation.

**Default**

### Operation Actions

Here is a sample of the please wait page.



Please wait while your query is being processed...

When the SQL query finishes, the please wait page disappears.



## Sample Customers



	▲ Employee # ▼	▲ First Name ▼	▲ MIDINIT ▼	▲ Last Name ▼	▲ WORKDEPT ▼
	000001	Erica	J	Piniero	A22
	000003	Laura	E	Klocke	D01
	000010	Ted	B	Cessna	G22
	000011	Paul	M	Thomas	R55
	000020	Steve	Q	Beechstreet	H22
	000030	John	C	Qu	G22
	000050	Paul	M	Tim	C01

## PossibleValues {}

PossibleValues can be used to change the default possible values behavior.

```
PossibleValues { fieldName:BasePath; copyList:BasePath,BranchType,BranchName;  
copyRule :usageId;}
```

Property	Value	Description
<i>fieldName</i>	field name	<i>This</i> property identifies which field in the main operation this configuration belongs to.

<i>copyList</i>	A list of comma separated field names	A list of fields to copy. Used for copying values from the possible values row to the main operation row.
<i>copyRule</i>	fieldNames (default), usageId	Tells how field values from the possible value row should be filled into/mapped to the actual row once a possible value choice is selected. Used for copying values from the possible values row to the main operation row.
optgroup	field name	Used for Possible Values Grouping.

See the Possible Values section for more details.

## ReportBreak {}

Reports are another important feature of WOW. Reports are used to perform different mathematical operations on the data in the table. Reports will find the minimum, maximum, sum, or average of any numeric data that is in the table. With a simple SQL command, the information in the table can be sorted out by specific groups such as work department or gender. The syntax is:

```
ReportBreak {}
```

In between the open and closed brackets, any of the following properties may be added. Property names must be followed by a colon, the property values should be separated with a comma, and end with a semicolon. For example:

```
ReportBreak { columnFunctions:max; columns:salary,comm;
breakColumns:workdept; overall:false;}
```

<b><u>Property</u></b>	<b><u>Value</u></b>	<b><u>Description</u></b>
columnFunctions	SUM   TOTAL   AVG   MIN   MAX   COUNT	The <b>columnFunctions</b> are simple mathematical commands such as SUM (or TOTAL), AVG, MIN, MAX, and COUNT.
columns	<i>field,field,...</i>   *	The name of the columns you wanted reports on. Each column is separated by a comma. If you wish to generate the report on every single column in the results, you may use an asterisk * instead of listing every column name.

breakColumns	<i>field,field,...</i>	Used to sort data by a specific field, such as work department, city, etc. This property is normally used in conjunction with the ORDER BY SQL command. Used to sort data by index. See below for an example. Where <i>func</i> in the property is replaced by the <b>columnFunctions</b> value. This property is used to change the text of the break row. For example, a SUM function would just say SUM in the break row. Assigning the breakText-SUM:Monthly Units; property would change that text to say Monthly Units.
breakCount	<i>field,field,...</i>	
breakText- <i>func</i>	<i>text</i>	
overall	TRUE   FALSE	Whether or not an overall "grand total" should be displayed at the bottom of the table. If you don't add the overall property group, it will automatically give an overall total. Setting overall to false is the only way to avoid displaying an overall total.
overallBreakText- <i>func</i>	<i>text</i>	Where <i>func</i> in the property is replaced by the <b>columnFunctions</b> value. This property is used to change the text of the overall break row. For example, a SUM function would just say SUM in the overall break row. Assigning the overallBreakText-SUM:Total Units; property would change that text to say Total Units.
reportSingleRow	TRUE   FALSE	Generate reports for a single row.



evenCSSStyle	<i>text</i>	<p>The name of the CSS style class applied to even report rows. The default value is <code>pjr-r-e</code> for normal report rows and <code>pjr-or-e</code> for overall report rows. If this report break property group is used for both normal and overall report breaks, then the style for overall report breaks cannot be altered from the default. To specify a style for overall report breaks, you need to create separate report break property groups.</p>
oddCSSStyle	<i>text</i>	<p>The name of the CSS style class applied to odd report rows. The default value is <code>pjr-r-e</code> for normal report rows and <code>pjr-or-e</code> for overall report rows. If this report break property group is used for both normal and overall report breaks, then the style for overall report breaks cannot be altered from the default. To specify a style for overall report breaks, you need to create separate report break property groups.</p>
evenBlankCSSStyle	<i>text</i>	<p>The name of the CSS style class applied to even blank report rows (a blank report row is used when a report row would normally be added, except that there is only one row over which to report). The default value is not to apply any type of special style.</p>
oddBlankCSSStyle	<i>text</i>	<p>The name of the CSS style class applied to odd blank report rows (a blank report row is used when a report row would normally be added, except that there is only one row over which to report). The default value is not to apply any type of special style.</p>

javaClass	<i>text</i>	The name of the Java class to use which provides report break functionality. This property should only be specified if you have created your own custom report break subclass.
-----------	-------------	--

## SignOn {}

This property group allows you to specify properties used in the sign-on process. It can either be specified for an application or an individual authentication operation (if specified in both places, the properties in the operation will take precedence).

Property	Value	Description
failureMessage	<i>text</i>	The message to display to the user when a sign-on attempt fails. The default is "Sign on failed. Please enter a valid user ID and password."
maxFailures	<i>integer</i>	The maximum number of times a user is allowed to fail the sign-on process. After this many sign-on failures, an application specific action takes place (the default action is to redirect to the original sign-on page). By default, there is no maximum number of failures.
auto	login   rememberMe	Activates auto <i>login</i> (auto-fills last entered user ID and password and logs user in automatically) or <i>remember me</i> (auto-fills the last entered user ID field).
cookiesMaxAge	integer (hours)	The length of time (in hours) the auto login/rememberMe information is retained in a cookie.
rememberFields	<i>field,field,...</i>	A list of fields to remember if using the "rememberMe" login feature.

## SpooledFile{}

[PRO] This property group controls how an operation's results will be exported out to a spooled file. For a spooled file export to be available, you must set the *spooledFile* property in the *TableDisplay* property group to "true". This property group can be specified in individual operations, or an application, or both. When an operation's results are exported, properties are taken first from the operation, and if not specified in the operation, from the

application. Property	Value	Description
align	<i>text</i>	The horizontal alignment for the data portion of the spooled file. This can either be "left", "right", or "center". The default value is right alignment.
colHeaderAlign	<i>text</i>	The horizontal alignment for the column headers in the spooled file. This can either be "left", "right", or "center". The default value is center alignment.
colHeaderSpacingBottom	<i>integer</i>	The number of empty spacing lines to put between the column headers and the first row of column data. The default value is one empty line.
colHeaderSpacingTop	<i>integer</i>	The number of empty spacing lines to put between the spooled file heading and the column headers. The default value is one empty line.
connectionAlias	<i>text</i>	The alias of the connection to use when exporting the spooled file to the iSeries. If this is not specified, then the operation's connection will be used. Whichever connection is used must be a connection to an iSeries, and not some other type of DB system.
displayColumns		The columns in the results which should be exported out to the spooled file. The default is all of the columns displayed in the HTML.
excludeColumns		The columns in the results which should not be exported out to the spooled file.
fileName	<i>text</i>	The name of the spooled file. If you leave this blank the iSeries will pick a default name for the spooled file.

generatorClass	<i>text</i>	The name of an optional Java class which is used to create the spooled file. The generator class must implement planetj.dataengine.spooledfile.ISpooledFileGenerator. If no generator class is specified, then the planetj.dataengine.spooledfile.DefaultSpooledFileGenerator class is used.
linesPerPage	<i>integer</i>	The number of lines per page in the spooled file. The default is 80.
outQueue	<i>text</i>	The output queue to which the spooled file will be exported. If you leave this blank the spooled file will be placed on the default output queue for the iSeries user which corresponds to the connection alias.
relativeWidths	<i>integer</i>	The relative widths of the columns in the spooled file. For example, if this property was set to "1,2,4,1" then the second column would be contain twice as many characters as the first and forth columns, and the third column would have twice as many as the second column.
spacingBottom	<i>integer</i>	The number of empty spacing lines at the bottom of the page. The default is none.
spacingLeft	<i>integer</i>	The number of empty spacing characters on the left of the page. The default is none.
spacingRight	<i>integer</i>	The number of empty spacing characters on the right of the page. The default is none.
spacingTop	<i>integer</i>	The number of empty spacing lines at the top of the page. The default is none.
userData	<i>text</i>	An optional informational tag for the spooled file. The value is automatically truncated to 10 chars.
width	<i>integer</i>	The maximum number of characters in each line of the spooled file. The default is 132.

## SQLContext {}

WOW can handle most complex SQL queries including ones where the table is dynamically selected. However, the WOW parser is not always able to determine the tables to use for field FD's. In these cases, the SQLContext property can be used to specify the appropriate tables from which to retrieve FD's.

**NOTE:** This replaces the 'tables' property in the StoredProcedure property group since SQLContext can be used for stored procedures too.

Property	Value	Description
tables	<i>library,table,...</i>	A list of tables to use for the reports field descriptors (e.g. planetj.customer, planetj.balancedta; ).

When prompting in such complex SQL queries, even though the tables have been specified using the SQLContext property group, WOW may still not know what FD's to use for each prompt. In these cases, use the FD parameter notation (e.g. ?1234 where the "1234" is the ID of the FD to be used or that prompt).

## StoredProcedure {}

This property group allows you to set the properties for a stored procedure call.

Property	Value	Description
rowCollection	TRUE   FALSE	Return row collection (result set) by the procedure.
successMessage	<i>text</i>	Completion message text.
tables	<i>library,table,...</i>	A list of tables to use for the reports field descriptors (e.g. planetj.customer, planetj.balancedta; ).

## Styles {}

This property group allows you to specify which CSS styles to use when generating an operation. Of course, any styles referred to in this property group must be defined in a .css files which is used by your application's theme.

Property	Value	Description
body	<i>css style name</i>	The general CSS style to apply to the body.
operation	<i>operation: text,...</i>	The style to use for the search operation. '=', '>=', and 'BETWEEN' are all examples of search operators. The following example removes the 'LIKE' operator and changes '=' to 'equals':

```
Styles {like: none; =: equals;}
```

searchLabel	css style name	The style to use for the search label.
submitButton	css style name	The style for the INPUT button used to submit the parameters the user has entered in.

## TableDisplay {}

There are many properties in the TableDisplay group. Most of them are all boolean values, unless specified otherwise (ex. selectionType and cellPadding). Boolean means they are either set to "true" or "false." If a property is set to true, the feature it controls will be visible to the user. If the property is set to false, the feature it controls will not be available to the user. The screen below will be used to demonstrate which icons will appear and disappear according to the boolean value.

SQL Operations

▲ Label ▼	▲ SQL Operation Type ▼	▲ Description ▼
<input type="radio"/> Possible values for WORKDEPT	Possible Values	
<input type="radio"/> Possible values for Jobs	Possible Values	
<input type="radio"/> Department Name Poss Values	Possible Values	View a sample database of emplo
<input type="radio"/> Departments	Association 1-1	View a sample database of emplo
<input type="radio"/> FoodPV	Possible Values	View a sample database of emplo
<input type="radio"/> Employees By Lastname	SQL	View a sample database of emplo

Insert

View

Edit

Copy

Delete

Property

Value

Description

buttons row

top | bottom

Designates where the buttons are located on the table display. The default is to show buttons on the bottom only. To shows buttons on both top and bottom, specify both values as follows: "top,bottom".

chart

TRUE | FALSE

Show the charting icon

cellPadding

integer

Padding between the border of a table cell and the contents of a table cell, specified in pixels.

colCnt	<i>integer</i>	Number of columns to generate for vertical generated row tables. Property only applies when display vertical is true. Number of columns to display when showing table results. Default is 2.
contextMenu	TRUE   FALSE	Whether or not the context menu is enabled for the table. Default is TRUE.
default row action	<i>action name</i>   NONE	The name of the action which should be performed when the row is double clicked. The default is to open the row for details view.
delete	TRUE   FALSE	Show the delete button. Default is false.
deleteAll	TRUE   FALSE	Show the deleteAll button. The default for this setting is false. Clicking this button deletes all the data being displayed.
deleteAllText	<i>text</i>	Text for the delete all button.
details	TRUE   FALSE	Show details button.
detailsText	<i>text</i>	Text for the details button.
displayVertical	TRUE   FALSE	Whether or not the table should be displayed vertically (false by default). If true, <b>colCnt</b> determines how many records are included per row.
double click	<i>text</i>	The name of the action to execute when the row is double clicked. By default, double clicking a row opens up the details screen for that row.
drawGrid	TRUE   FALSE	Show grid lines. The grids are the vertical and horizontal lines that separate the rows and columns. The default value is true. Grid lines tend to improve the look and feel of the table being displayed.
edit	TRUE   FALSE	Show the Edit Record button.
editFD	TRUE   FALSE	Show the red edit FD. The default value is false. Clicking this button allows you to edit the Field Descriptors for the displayed data.
editText	<i>text</i>	Text for the edit button.

excel	TRUE   FALSE	Show the Excel icon. The default value is true. Clicking this icon sends the selected data into a Microsoft® Excel spreadsheet.
excelXls	TRUE   FALSE	Show Excel file button.
forceRefresh	TRUE   FALSE	Force refresh of data before displaying. The default is false.
header	TRUE   FALSE	Show the column header.
helpTextInHeaders	ALL   NONE   <i>field,field,...</i>	This is a list of the columns which will display the hover help text defined in the field descriptor when the user hovers over the column header. You can also use the special values ALL or NONE to refer to all columns in the table. The default value is ALL, so by default, all column headers will use their hover help text.
helpTextInCells	ALL   NONE   <i>field,field,...</i>	This is a list of the columns which will display the hover help text defined in the field descriptor when the user hovers over the column cells. You can also use the special values ALL or NONE to refer to all columns in the table. The default value is ALL, so by default, all column cells will use their hover help text. For large tables, you can reduce the amount of HTML which is generated by disabling the hover help text for cells.
highlight style	<i>css class</i>   NONE	The name of the CSS class which is applied to the row when the cursor is hovering over the row. The default value is "pjc-highlight".
insert	TRUE   FALSE	Show the insert button.
insertable	TRUE   FALSE	Determines whether or not the table should allow direct inserts without viewing the details of a single row.
insertText	<i>text</i>	Text for the insert button.
linkable	TRUE   FALSE	Determines whether or not the user has the option to generate an HTML link directly to the current results.



msAccess	TRUE   FALSE	Show the Microsoft Access quick link.
msWord	TRUE   FALSE	Show the Microsoft Word quick link. The default value is true. Clicking this icon sends the selected data into a Microsoft Word document.
multipleDelete	TRUE   FALSE	Determines whether or not deleting multiple rows is supported.
pdf	TRUE   FALSE	Show the PDF quick link.
print	TRUE   FALSE	Show the print quick link. The default value for this is true. Clicking this icon displays the selected data in a printer-friendly format.
refresh	TRUE   FALSE	Show the refresh pinwheel quick link. The default value for this is true. The refresh button allows you to refresh the data being displayed, much like the refresh button on your web browser.
removeAll	TRUE   FALSE	Show the remove all button.
rowCopy	TRUE   FALSE	Text for the row copy button.
rowCopyText	<i>text</i>	Text for the row copy button.
row select sound	<i>text</i>	The URL of a sound file. This sound file will be played when the user selects the row.
selection style	<i>css class</i>   NONE	The name of the CSS class which is applied to the row which is currently selected (by a single click from the user). The default value is "pjc-selection".
selectionType	NONE   SINGLE   MULTIPLE	Indicates how the data in the table can be selected. MULTIPLE allows the user to select more than one entry in the table at a time with check boxes. NONE eliminates the option of selecting specific entries from the table. The default value is MULTIPLE.
selectableRecords	TRUE   FALSE	Determines whether or not the records in the Table can be selected.
showSelection	TRUE   FALSE	Shows the selection buttons for each record.

single click	<i>text</i>	The name of the action to execute when the row is single clicked. By default, single clicking the row does not execute any actions. This property cannot be specified if the double click property is also specified.
sorting	TRUE   FALSE	Showing descending and ascending sort buttons next to column headers. The default value is true. Sorting allows you to sort each column by alphabetical or numeric order.
spooledFile	TRUE   FALSE	Show the export to spooled file link.
tableClass	<i>subclass</i>	You can optionally specify a subclass of HTMLTable to use when rendering your RowCollection.
tableWidth	<i>integer</i>	Specify width of results table.
updateable	TRUE   FALSE	Allow each field to be updated by the user directly from the displayed table. The default value is false. By changing this value to true, you will have the ability to edit each entry directly from the table shown.
updateText	<i>text</i>	Text for the update button.
wrapHeaders	TRUE   FALSE	Do not allow wrap.
xml	TRUE   FALSE	Show XML quick link. The default value is true. By clicking this icon, WOW will send the selected data into an XML document. An XML ready browser is required for this option.

## Tabs {}

This property group allows you to configure tabs (display the results of an operation in a tabbed layout).

Property	Value	Description
allowInTab	RESULTS   DETAILS   BOTH   NEVER	Determines what can be displayed inside of this tab.
automaticTabView	TRUE   FALSE	Automatically show the tabbed view of a query result.
defaultTab	<i>tab field name</i>	Default tab to display.
emptyMessage	<i>text</i>	Message displayed to the user when there are no results returned. Use in conjunction with the <b>hideWhenEmpty</b> property.

hideWhenEmpty	TRUE   FALSE	Determines whether or not to display an empty row collection if there are no results returned to the tabbed operation. Use in conjunction with the <b>emptyMessage</b> property.
tabFields	<i>tab field name,...</i>	Specifies which fields are to be rendered as tabs.
tabFieldsExclude	<i>tab field name,...</i>	Specifies which fields are not to be rendered as tabs.
tabHeadingsJSP	<i>file path</i>	The JSP to display the tab's headings.
tabParentJSP	<i>file path</i>	The JSP to display the tab's parent row.
maxTabsPerLine	<i>integer</i>	The maximum number of tabs that can be displayed in a single line on the screen (default is 10).
alwaysShowSearch	TRUE   FALSE	Hide search parameters once results for the parent tab are returned.

## XLS {}

[EE] WOW can export real time data to an existing Excel spreadsheet. The existing spreadsheet may have macros, graphs, charts and other items predefined. This property group allows you to set properties for an Excel worksheet.

Property	Value	Description
sheetIndex	<i>integer</i>	Index of Excel worksheet (starting with 1).
sheetName	<i>text</i>	Name to give new Excel worksheet.
xmlFormat	TRUE   FALSE	Forces Excel export to use the openxmlformat for spreadsheets (xlsx). This allows an export to contain more than the restricted 65536 rows.

## Sorting

When a query includes an ORDER BY clause, the results of the query will be displayed in the specified order. The column or columns used in the ORDER BY clause are indicated in the results by highlighted arrows. For example, these results:

▲ ID ▼	▲ Name ▼	▲ Balance ▼	▲ Level ▼	▲ State ▼
85	Jess	338	Basic	Arizona
87	Trent	138	Basic	Arizona
89	Kelly	662	Basic	Arizona
90	Manny	565	Basic	Arkansas
91	Vincent	335	Basic	Arkansas
92	Hidalgo	449	Basic	Arkansas
94	Muade	679	Basic	Arkansas
95	Ethan	712	Basic	Arkansas
96	Edna	272	Basic	Arkansas
97	James	433	Basic	Arkansas
98	Phil	237	Basic	Arkansas
99	Tony	595	Basic	Arkansas
100	Mandy	213	Basic	Arkansas
101	Eunice	343	Basic	Arkansas
102	Sandra	475	Basic	Arkansas
103	Tori	238	Basic	Arkansas
104	Shelly	131	Basic	Arkansas
105	Nikki	565	Basic	Arkansas
106	Brian	363	Basic	Arkansas
88	Molly	232	Bronze	Arizona
84	Javo	228	Gold	Arizona
86	Stone	1448	Gold	Arizona
107	Theo	227	Gold	Arkansas
93	Tess	559	Platinum	Arkansas

were produced by a query which contained ORDER BY LEVEL in its ORDER BY clause. The highlighted arrow in the LEVEL column indicates to the user that the results are sorted in ascending order by the LEVEL column.

When the user clicks on one of the sort arrows in a column, the results are resorted first by using the column which was clicked on, and then by any columns by which the results were previously sorted. So in the above example, if the user clicked on the down arrow in the STATE column, the results would be sorted primarily in descending order by STATE, and then in ascending order by LEVEL:

▲ ID ▼	▲ Name ▼	▲ Balance ▼	▲ Level ▼	▲ State ▼
90	Manny	565	Basic	Arkansas
91	Vincent	335	Basic	Arkansas
92	Hidalgo	449	Basic	Arkansas
94	Muade	679	Basic	Arkansas
95	Ethan	712	Basic	Arkansas
96	Edna	272	Basic	Arkansas
97	James	433	Basic	Arkansas
98	Phil	237	Basic	Arkansas
99	Tony	595	Basic	Arkansas
100	Mandy	213	Basic	Arkansas
101	Eunice	343	Basic	Arkansas
102	Sandra	475	Basic	Arkansas
103	Tori	238	Basic	Arkansas
104	Shelly	131	Basic	Arkansas
105	Nikki	565	Basic	Arkansas
106	Brian	363	Basic	Arkansas
107	Theo	227	Gold	Arkansas
93	Tess	559	Platinum	Arkansas
85	Jess	338	Basic	Arizona
87	Trent	138	Basic	Arizona
89	Kelly	662	Basic	Arizona
88	Molly	232	Bronze	Arizona
84	Javo	228	Gold	Arizona
86	Stone	1448	Gold	Arizona

**NOTE:** The sort arrows in both the STATE and LEVEL columns are highlighted, since both of these columns are used in sorting the results.

By default, all database columns are sortable, and all derived columns are not sortable. To change the default behavior, you can edit the *sortable* property for a column in that column's field descriptor.

## Controlling the Sorting Behavior

This section describes what you can do to control how WOW displays sorting columns to the user.

### Changing the Column Heading

In this example from earlier in the chapter, we have results which are sorted by two columns, STATE and LEVEL:

▲ ID ▼	▲ Name ▼	▲ Balance ▼	▲ Level ▼	▲ State ▼
90	Manny	565	Basic	Arkansas
91	Vincent	335	Basic	Arkansas
92	Hidalgo	449	Basic	Arkansas
94	Muade	679	Basic	Arkansas
95	Ethan	712	Basic	Arkansas
96	Edna	272	Basic	Arkansas
97	James	433	Basic	Arkansas
98	Phil	237	Basic	Arkansas
99	Tony	595	Basic	Arkansas
100	Mandy	213	Basic	Arkansas
101	Eunice	343	Basic	Arkansas
102	Sandra	475	Basic	Arkansas
103	Tori	238	Basic	Arkansas
104	Shelly	131	Basic	Arkansas
105	Nikki	565	Basic	Arkansas
106	Brian	363	Basic	Arkansas
107	Theo	227	Gold	Arkansas
93	Tess	559	Platinum	Arkansas
85	Jess	338	Basic	Arizona
87	Trent	138	Basic	Arizona
89	Kelly	662	Basic	Arizona
88	Molly	232	Bronze	Arizona
84	Javo	228	Gold	Arizona
86	Stone	1448	Gold	Arizona

The results are sorted first by state, and then by level .

The highlighted arrows in the column headers let the user know which columns are used to sort the results, but it is not possible to determine the order in which the columns were used to sort the results without closely examining the values in those two columns. However, it is possible to have WOW alter the column header to show the sort order as well as the column name.

For example these results:

▲ ID ▼	▲ Name ▼	▲ Balance ▼	▲ 2-Level ▼	▲ 1-State ▼
90	Manny	565	Basic	Arkansas
91	Vincent	335	Basic	Arkansas
92	Hidalgo	449	Basic	Arkansas
94	Muade	679	Basic	Arkansas
95	Ethan	712	Basic	Arkansas
96	Edna	272	Basic	Arkansas
97	James	433	Basic	Arkansas
98	Phil	237	Basic	Arkansas
99	Tony	595	Basic	Arkansas
100	Mandy	213	Basic	Arkansas
101	Eunice	343	Basic	Arkansas
102	Sandra	475	Basic	Arkansas
103	Tori	238	Basic	Arkansas
104	Shelly	131	Basic	Arkansas
105	Nikki	565	Basic	Arkansas
106	Brian	363	Basic	Arkansas
107	Theo	227	Gold	Arkansas
93	Tess	559	Platinum	Arkansas
85	Jess	338	Basic	Arizona
87	Trent	138	Basic	Arizona
89	Kelly	662	Basic	Arizona
88	Molly	232	Bronze	Arizona
84	Javo	228	Gold	Arizona
86	Stone	1448	Gold	Arizona

are sorted first by STATE and then by LEVEL, which can be immediately seen by looking at the column names.

Controlling the column names is done with using the *heading* property of the Sorting property group. The value specified in the *heading* property will be shown as the column name for columns which are used to sort the results. The special placeholders *%name* and *%sortindex* will be replaced with the column name and sorting index respectively. So in the above example this Sorting property group was used:

```
Sorting {
  heading: %sortindex-%name;
}
```

If you wanted the sort index to be displayed after the column name with no hyphen, then the property group would look like this:

```
Sorting {
  heading: %name %sortindex;
}
```

The Sorting property group can be specified in an Operation, or in an Application (in which case it will apply to all Operations in that Application).

## Changing the Header Style

The *css* property in the Sorting property group can be used to set a different CSS style on column headers used for sorting. For example, if this is the sorting property group

```
Sorting {
  heading: %sortindex-%name;
  css: sort;
```

























}

Then the CSS "sort" class will be applied to all the headers of columns which are used to sort the results. (The column headers are TD HTML elements.) In addition, if the LEVEL column is the first sort column, the "sort\_LEVEL" and "sort\_1" CSS classes are also applied to the header. When STATE is the 2nd sort column, the "sort\_STATE" and "sort\_2" classes will be applied to that header. This allows different sorting columns to be given different styles, based either on that column's name, or sorting index.

If the following CSS classes are defined:

```
.sort.sort_1 {  
background-color: cyan;  
}  
.sort.sort_2 {  
background-color: lime;  
}
```

Then the header of the first sorting column will be cyan, and the header of the second sorting column will be lime:

	▲ ID ▼	▲ Name ▼	▲ Balance ▼	▲ 2-Level ▼	▲ 1-State ▼
	90	Manny	565	Basic	Arkansas
	91	Vincent	335	Basic	Arkansas
	92	Hidalgo	449	Basic	Arkansas
	94	Muade	679	Basic	Arkansas
	95	Ethan	712	Basic	Arkansas
	96	Edna	272	Basic	Arkansas
	97	James	433	Basic	Arkansas
	98	Phil	237	Basic	Arkansas
	99	Tony	595	Basic	Arkansas
	100	Mandy	213	Basic	Arkansas
	101	Eunice	343	Basic	Arkansas
	102	Sandra	475	Basic	Arkansas
	103	Tori	238	Basic	Arkansas
	104	Shelly	131	Basic	Arkansas
	105	Nikki	565	Basic	Arkansas
	106	Brian	363	Basic	Arkansas
	107	Theo	227	Gold	Arkansas
	93	Tess	559	Platinum	Arkansas
	85	Jess	338	Basic	Arizona
	87	Trent	138	Basic	Arizona
	89	Kelly	662	Basic	Arizona
	88	Molly	232	Bronze	Arizona
	84	Javo	228	Gold	Arizona
	86	Stone	1448	Gold	Arizona

Using different colors for the sorting columns is especially useful when report breaks are displayed in the results.



## Associations

Associative programming is one of the key features of WOW. An association links data from two different tables by using fields that are common for both tables. An association may also link data from a table to some other functionality. There are SQL, HTML, and Java associations.

In the example below, we will link the EMPLOYEE table with the DEPARTMENT table which can both be found in the PJDATA schema. Any two tables can be linked together as long as they have data that is similar or linkable, and the tables are accessible through a previously created database connection. There are two types of SQL associations that can be used with WOW; they are 1-1 Association and 1-Many Association. After an association is created a hyperlink will be available for the user to click on. The screenshot below is an example of this.

▲ Last Name ▼ 	▲ Work Dept # ▼ 	▲ Phone # ▼ 
TstLNM	<a href="#">A00</a>	3978
THOMPSON	<a href="#">B01</a>	3476
KWAN	<a href="#">C01</a>	4738
GEYER	<a href="#">E01</a>	6789
STERN	<a href="#">E21</a>	6423
PULASKI	<a href="#">D21</a>	7831
HENDERSON	<a href="#">E11</a>	5498
SPENSER	<a href="#">E21</a>	0972
LUCCHESI	<a href="#">A00</a>	3490
O'CONNELL	<a href="#">A00</a>	2167
QUINTANA	<a href="#">C01</a>	4578
NICHOLLS	<a href="#">C01</a>	1793
ADAMSON	<a href="#">D11</a>	4510

Below is a brief explanation of the different kinds of association, such as 1-1 Associations and 1-Many Associations.

### 1-1 Association

A 1-1 Association links a specific field in a table to a single entry. The format is similar to viewing an entry using the view button described in the introduction. Below is an example of what to expect after creating a 1-1 Association and following the hyperlink that was created.

Fields marked with an asterisk (\*) are required.

Cancel

Department Number: A00      Department Name: SPIFFY COMPUTER SERVICE DIV.  
 Manager Number: 000010      Admr. Dept: A00  
 Location:

Cancel

## 1-Many Association

A 1-Many association is the same as a 1-1 association except the 1-Many association will link you to more than a single row of data. 1-Many associations are useful when there is more than one row of data you would like displayed. Below is an example of what to expect from a 1-Many association (notice that it links you to more than one data record as opposed to the 1-1 Association linking to a single view only entry).



	▲ Employee # ▼	▲ First Name ▼	▲ Middle Initial ▼	▲ Last Name ▼	▲ V
	<a href="#">000030</a>	SALLY	A	KWAN	<a href="#">C01</a>
	<a href="#">000130</a>	DELORES	M	QUINTANA	<a href="#">C01</a>
	<a href="#">000140</a>	HEATHER	A	NICHOLLS	<a href="#">C01</a>
	<a href="#">200140</a>	KIM	N	NATZ	<a href="#">C01</a>
	<a href="#">8887</a>	Joe	b	Public	<a href="#">C01</a>

Insert

## HTML Code Association

Essentially an HTML Code operation with association capabilities, this type of association allows you to link to some specified HTML. This is an exceptionally powerful feature in WOW and is often used for stylizing reports and other data.

## Full Field Rendering

WOW 6.45 and later includes enhanced support for HTML Code Association scripting. New support includes the following features, which are coded directly into the HTML Code Association. The special character of "\*" appended as the last character indicates that WOW should generate the entire field rendering, not just the value. For example, a field with an association referenced with ??Field will only display the value and not a hyperlink-capable rendering. However ??Field\* would render the entire hyperlink HTML code.

WOW Script

Description

??FLDNAME\*

Using the associated Row, render the entire fields formatting as defined in its field descriptor.

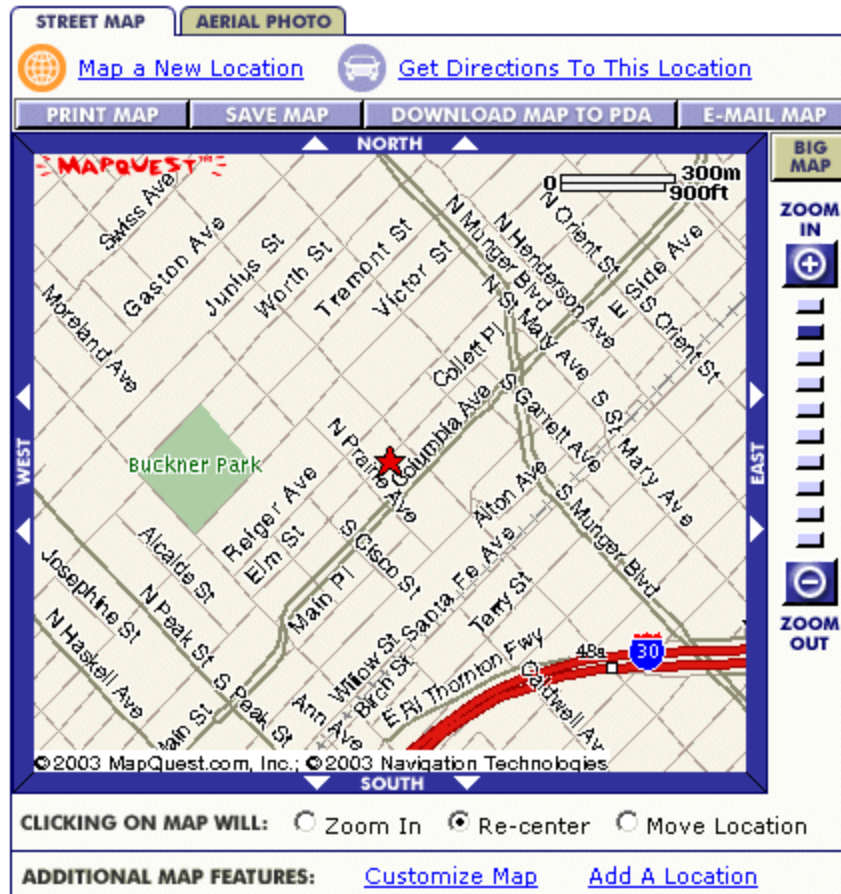
**NOTE:** In all cases, the fields can only be used for display and not for updating databases.  
**HTML Reference Association**

In this association, rather than linking the data between two tables, records, or Rows, it is linking the current data with some HTML reference. For instance, take the example listed below. The results have a bunch of Rows with address information. Each Row also contains a derived field that has its association set a HTML Reference Association that links to MapQuest®.

▲ Street ▼	▲ City ▼	▲ State ▼	▲ Zip code ▼	
4859 Elm Ave	Dallas	TX	75217	<a href="#">View Map</a>
21B NW 135 St	Clay	NY	13041	<a href="#">View Map</a>
PO Box 79	Broton	VT	5046	<a href="#">View Map</a>
3 Alpine Way	Helen	GA	30545	<a href="#">View Map</a>

Clicking the "View Map" link on the first record would bring up the following:

4859 Elm Aly  
Dallas, TX  
75246-1274, US



## Associated Java Operation

[PRO] Actual calls to Java methods can be executed via an Associated Java Operation. These methods must be static and all of their parameters must be of type `java.lang.String`, with the exception of a few special cases listed below. This operation has specific signature that is used to accomplish this task. The name of the class, name of the method to be called, and the parameters to the method are separated by the "pipe" special character which is designated as the vertical bar '|'. The first part of the operation is the fully qualified class name of the class that the method is to be executed on. The second part of the operation is the name of the method that will be called on the fully qualified class. This method must be static since there will not be a specific instance of the fully qualified class. Every part thereafter is treated as a String parameter to the method.

For example, if we have a class `planetj.examples.Log` that has method `writeEntry` which takes an `entry` argument that writes an entry to a log located on the file system, it would be called in the following manner:

```
planetj.examples.Log|writeEntry|Calling Java method from an operation
```

This would result in the method `writeEntry` in the class `planetj.examples.Log` to be executed with the String argument of "Calling Java method from an Operation."

There are certain parameters that can be specified that will automatically be filled in with their associated values.

Parameter	Description
*REQUEST	Passes the current Request Object to the method.
*RESPONSE	Passes the current Request Object to the method.
*USER	Passes the current User Object to the method.
*ROW	Passes the current Row Object to the method.
*ROW_COLLECTION	Passes the current Row Collection Object to the method.

For example, if we have a class `planetj.examples.Log` that has a method `logParameterValues` which takes an `HttpServletRequest` object that writes all of the current parameters on that request to the file system, it would be called in the following manner:

```
planetj.examples.Log|logParameterValues|*REQUEST
```

The Associated Java Operation also allows for dynamic entries from the current Row that is associated with the Operation. For example, if we have an Associated Java Operation that has the Make, Model, and Year of vehicles and the names of the columns in this row are specifically "MAKE", "MODEL", and "YEAR", these values can be passed to a Java method in the following manner:

```
planetj.examples.Log|logCarMakeModelYear|??MAKE¿|??MODEL¿|??YEAR¿
```

The dynamic entries must be designated by start and end characters in order for WOW to determine the beginning and end of the column name. The start characters are '??' and the corresponding ending character is "¿" (this character can be typed by using ALT+0191).

## Creating Associations

Creating an association is very similar to creating any other type of Operation. The first thing you need to do is create an Operation. To create an association you change the Operation type from SQL to one of the association operation types. Association operations have the word "Association" in their display name. Then, you just set its operation code. After the operation is set, then you need to modify a field's Field Descriptor to set the association, so when the Field generates, it will have a link to the association. The two examples below show how to create both SQL and HTML associations.

### SQL Association Example

For an SQL association, the operation type should be either a 1-1 Association or a 1-Many Association. The screenshot below shows an example an SQL 1-Many association:

Basic			
Label*	<input type="text" value="Employees"/>	Title:	<input type="text" value="Sample Employees"/>
Type*	<input type="text" value="Association 1-MANY"/>	Description:	<input type="text" value="View a sample database of employees"/>
Operation Code:	<input type="text" value="SELECT * FROM SAMPLE.EMPLOYEE"/>		
Instructions:	<input type="text"/>	Application:	<a href="#">View Application</a>

The Type and Operation Code are the two most commonly used fields when creating an association. The code used to create an association may vary depending upon the type of association you are creating (for instance, HTML Associations are different from SQL associations). The screenshot below shows an SQL association. You only need to pay attention to the Operation Code. The code shown will link the DEPARTMENT table to the EMPLOYEE table using the similar fields WORKDEPT and DEPTNO:

Basic			
Label*	<input type="text" value="WorkDept Assoc"/>	Title:	<input type="text"/>
Type*	<input type="text" value="Association 1-MANY"/>	Description:	<input type="text" value="dept to workdept assoc"/>
Operation Code:	<input type="text" value="SELECT * FROM pjdata.employee where workdept = ??deptno"/>		
Instructions:	<input type="text"/>	Application:	<a href="#">View Application</a>

The operation code used for this association is:

```
SELECT * FROM pjdata.employee WHERE workdept = ??deptno
```

Notice the SQL code is similar to a SELECT SQL statement. The first thing you need to notice is the table it is selecting from. This table contains the information which we will link *to*. Next is the WHERE statement, this statement shows which field the association is being linked *from*, in this example the WORKDEPT field in EMPLOYEE is being linked with the DEPTNO field which is located in the DEPARTMENT table. The linking of the two fields is done by using an equals (=) sign followed by double question marks (??) and the field the association will be linked *from*.

The DEPARTMENT table is not mentioned anywhere in the code because the association link will be visible in any query on the DEPARTMENT table. After inserting the Association you will see it listed in the group of other Operations that you have created for your application. The final step to creating an association is to assign your association to a specific field. To do this run an Operation to display the table you are using for your association; in the example above, we are using the DEPARTMENT table so we will run the Operation to display the DEPARTMENT table. Your query should look similar to the screenshot below, substituting the table you are using with the DEPARTMENT table:

Departments				
	▲ Department # ▼	▲ Department Name ▼	▲ Manager # ▼	▲ Administrator Dept ▼
	A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
	B01	PLANNING	000020	A00
	C01	INFORMATION CENTER	000030	A00
	D01	DEVELOPMENT CENTER		A00
	D11	MANUFACTURING SYSTEMS	000060	D01
	D21	ADMINISTRATION SYSTEMS	000070	D01
	E01	SUPPORT SERVICES	000050	A00
	E11	OPERATIONS	000090	E01
	E21	SOFTWARE SUPPORT	000100	E01
	F22	BRANCH OFFICE F2		E01
	G22	BRANCH OFFICE G2		E01
	H22	BRANCH OFFICE H2		E01
	I22	BRANCH OFFICE I2		E01
	J22	BRANCH OFFICE J2		E01

Insert

Once you have a screen similar to the one above, but without the associations, you can set up the association you previously created. To do this click on the gear icon next to the column you want to use along with your association. In this example, we will click the gear icon directly to the right of the DEPTNO column as shown below:

	▲ DEPTNO ▼	▲ DEPTNAME ▼	▲ MGRNO ▼	▲ ADMRDEPT ▼
	A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
	B01	PLANNING	000020	A00
	C01	INFORMATION Consulting	000030	A00
	D01	DEVELOPMENT CENTER		A00
	D11	MANUFACTURING SYSTEMS	000060	D01
	D21	ADMINISTRATION SYSTEMS	000070	D01
	E01	SUPPORT SERVICES	000050	A00
	E11	OPERATIONS	000090	E01
	E21	SOFTWARE SUPPORT	000100	E01
	F22	BRANCH OFFICE F2		E01

The 'gear' icon located next to each column is used to edit the Field Descriptors of each field. For now, all you will have to do is locate the association operation field which is found under the Advanced Settings section.

To activate the newly created association, pick the name that you gave your newly created association operation. In this example, we will pick the "WorkDept Assoc" operation as shown below:

Advanced Settings

☒ Enter Class Name:
 

Field Class:

☐ Select Existing:
 

Address 1

Field Descriptor Type \* : Default

Concurrency \* : Concurrent Updates and Deletes Allowed

Remarks:

Association Operation: WorkDeptAssoc

Formatter Class: (None)

Getter Method:

Setter Method:

XML Tag:

Once you have saved your change to the field descriptor, the association is complete. Now, whenever the DEPTNO field of the DEPARTMENT table is displayed, it will have a hyperlink to the employees associated with that department as shown below.

## HTML Code Association Example

In this example, we will demonstrate how the HTML Code Association can be used to easily arrange and format data. In particular, we are going to be creating simple, dynamic PlanetJ business cards. In other words, the user will click 'Generate Business Card' and WOW will use row parameters (??FIELD) to dynamically plug in data to a HTML based business card template. The goal here is to show you how this association type can be used to format your data in just about any way imaginable.

### Overview

We want to transform our employee data from the standard table layout into a nice, stylized business card layout.

First Name	Last Name	Position	Company	Phone	Email	Business Card
Erica	Piniero	DIRECTOR	PlanetJ Corporation	760.432.0600	epiniero@planetjavainc.com	<a href="#">Generate Business Card</a>
Laura	Klocke	ANALYST	PlanetJ Corporation	760.432.0600	lklocke@planetjavainc.com	<a href="#">Generate Business Card</a>
Ted	Cessna	MARKETER	PlanetJ Corporation	760.432.0600	tcessna@planetjavainc.com	<a href="#">Generate Business Card</a>
Paul	Thomas	MANAGER	PlanetJ Corporation	760.432.0600	pthomas@planetjavainc.com	<a href="#">Generate Business Card</a>





## Create Employee Operation

First, we need to create the operation that will return the data and derived field that on which we set the association to. Insert a new operation of type 'SQL Operation' and enter the following Operation Code:

```
SELECT *, 'Generate Business Card' AS busCard FROM PJDATA.EMPLOYEE
```

## Create HTML Code Association Operation

Second, we will create the HTML Code Association that will act as the HTML template for the business card. Insert a new operation of type 'HTML Code Association' and enter the HTML given below in the Operation Code field. The [blue](#) text is all standard HTML and CSS and, if you are not too familiar with either, can easily be generated using an HTML editor such as Adobe® Dreamweaver® or Microsoft® FrontPage®. The important code to note is the flagged by [red](#) text that contains new parameters used to retrieve data from the data row. These are in the form: `??FIELDNAME`.

**NOTE:** There must always be a space after a row parameter.

```
<div style="width: 340px; height: 196px; background-image: url(user/sample/
images/PJ_BusinessCard.jpg); background-repeat: no-repeat;">

<!-- Name, Position, Company -->
<div style="position: relative; text-align: right; font-family:
Arial,Helvetica,sans-serif; left: 130px; top: 30px; width: 189px; height:
56px;">
<span style="color: #666666; font-size: 16px; font-weight: bold;">
??firstname ??lastname
</span><br />
<span style="color: #333333; font-size: 14px; font-weight: bold;">
??position
</span><br />
<span style="color: #660000; font-size: 14px; font-weight: bold;">
??company
</span>
</div>

<!-- Telephone, Email -->
<div style="position: relative; text-align: right; font-family:
Arial,Helvetica,sans-serif; left: 82px; top: 81px; width: 189px; height:
29px;">
<span style="color: #999999; font-size: 10px; font-weight: bold;">
??telephone
</span><br />
<span style="color: #999999; font-size: 10px; font-weight: bold;">
??email
</span>
</div>
</div>
```

This HTML code is simply laying our employee data (the row parameters) on top of a background image.



## Set the Association to a Field

Third, and last, we need to assign the association we created in step 2 to the derived `busCard` field from step 1. Create a derived field descriptor for the `busCard` field in the `pjdata.employee` table, set its Association Operation to the one created in step 2, and update.

That's it! All that is left to do is run the application and click the 'Generate Business Card' field. Hopefully, you have seen from this example that the layout of your data is only limited to what you can create using HTML and CSS. Invoices, reports, dynamic web pages, etc. are all as easy as plugging in `??FIELDNAME`.

This example only used the basic row parameter. However, by using row parameters with Full Field Rendering notation, you can generate fields within an HTML Code Association with all the formatting (possible values, association hyperlinks, etc.) specified in their respective field descriptors rather than just plain field value.

## HTML Reference Association Example

This example will show how to create an HTML Reference Association. We will create an association which links from a row containing address information to a map of this address. MapQuest will provide the actual maps; all our association has to do is pass MapQuest the address information. First, an SQL Operation needs to be created to select the address information. The screenshot below shows part of the results from an address file.

▲ Street ▼	▲ City ▼	▲ State ▼	▲ Zip code ▼	
4859 Elm Ave	Dallas	TX	75217	<a href="#">View Map</a>
21B NW 135 St	Clay	NY	13041	<a href="#">View Map</a>
PO Box 79	Broton	VT	5046	<a href="#">View Map</a>
2 Almine Way	Helen	GA	30545	<a href="#">View Map</a>

You'll notice that it contains a derived field that has a link to view map for each address. We'll get to setting the association, but first, we need to create the HTML Reference Association. Create an Operation and set its type to HTML Reference Association. For its operations code, enter the following URL:

```
http://www.mapquest.com/maps/map.adp?address=??street&city=??city&state=??state&zipcode=??zipcod&zoom=8
```

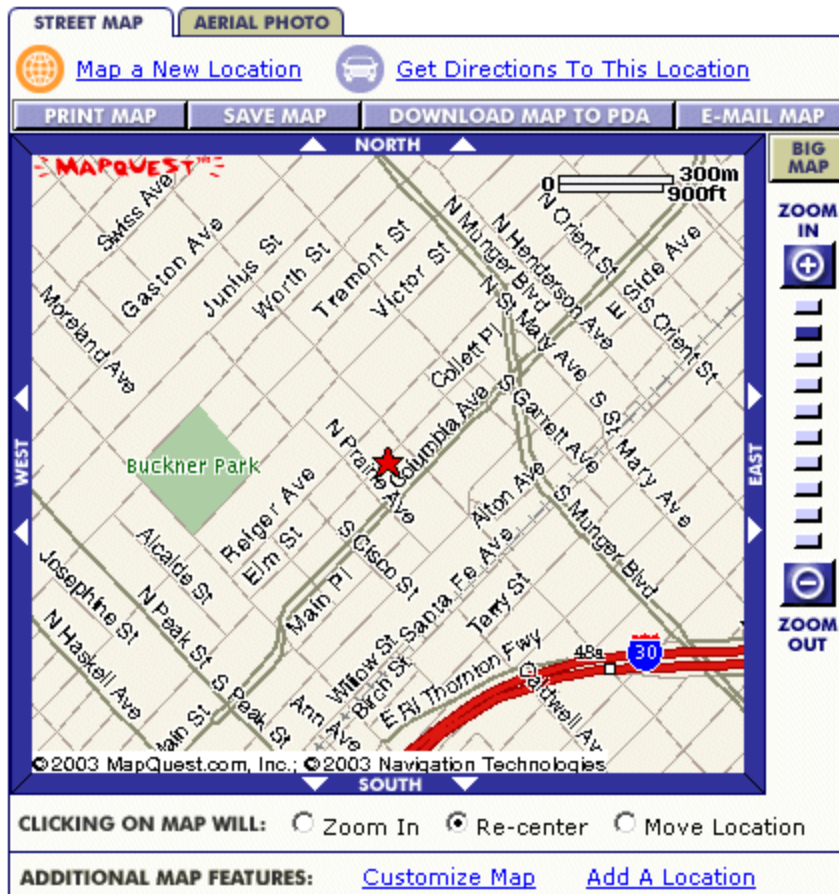
## Basic

Label*:	<input type="text" value="View Map"/>	Title:	<input type="text" value="Links an associated address"/>
Operation Type*:	<input type="text" value="HTML Reference Association"/>	Description:	<input type="text" value="This association can be used to link an address to a map."/>
Operation Code:	<input type="text" value="http://www.mapquest.com/maps/map.adp?address=??street&amp;city=??city&amp;state=??state&amp;zipcode=??zipcode&amp;zoom=8"/>		

With all operation code strings, you can specify parameters. In the above URL, there are parameters specified for the street, city, state, and zip code. That way, when the association is set on a Field, the link to the association will actually have the parameter values set from its Row's values. The above results contain the columns: street, city, state, and zip code. So when the URL link is generated, any Row parameters are replaced with the Field's value. The following is the link generated when for the first record, which when clicked opens up the following map.

<http://www.mapquest.com/maps/map.adp?address=8959+Elm+Ave&city=Dallas&state=TX&zipcode=75217&zoom=8>

**4859 Elm Aly**  
**Dallas, TX**  
**75246-1274, US**



In order for the link to show up in the results, the association needs to be set on a Field.

In this example a derived field is created. A derived field isn't absolutely necessary. The association could have been set on the street field, in which case its display value would be a link to the map.

Open up the row Manager and click the edit icon next to the FieldDescriptor for the Field you wish to have the associated map link generated for. Then change the FieldDescriptor's association operation to the newly create HTML Reference Association.

Advanced Settings		
Field Class:	<input checked="" type="radio"/> Enter Class Name: <input type="text"/>	<input type="radio"/> Select Existing: <input type="text" value="Address 1"/>
Field Descriptor Type:	<input type="text" value="Derived"/>	Formatter Class: <input type="text" value="- None -"/>
Concurrency:	<input type="text" value="Concurrent Updates and Deletes Allowed"/>	Getter Method: <input type="text"/>
Remarks:	<input type="text"/>	Setter Method: <input type="text"/>
Association Operation:	<input type="text" value="View Map"/>	XML Tag: <input type="text"/>

By default the HTML Reference Association will open a different window as your application which in many cases is desired, but in some cases you may want to run the link in the same window. For this case you need to edit the HTML Reference Association operation and in the properties section add the property group Browser with the property target set to `_self`.

```
Browser {target:_self; }
```

## Associated Inserts

[EE] An Associated Insert will insert a row or collection of rows into the database, using one or more values from a row in an associated table. It is possible to insert a row where some of the row's values are dynamically entered by the user and other values are retrieved from a row in an associated table.

Creating an Associated Insert is very similar to doing basic 1-1 or 1-many associations. To create an association, you change the Operation Type from SQL to either the 1-1 Association or 1-Many Association Type. Then you need to set its operation code. After the operation code has been set, you need to modify a Field's Field Descriptor to set the association, so when the field is generated it will have a link to insert in its associated row or rows. This process is described in detail below:

## SQL Associated Insert Example

To create an associated insert, select the Create Operation link from the TOC, and 1-Many for Operation Type. You next have to enter the Operation Code for the associated insert. This is very similar to the code for a normal SQL insert, except that you must specify where to retrieve the associated value for the insert from. For each associated value you wish to

insert, you use two question marks followed by the name of the column containing the data in the associated table (not the table where the row is being inserted).

The Code shown below will link the Department table to the Employee table allowing you to insert into the Employee table using the similar fields WORKDEPT and DEPTNO:



### Applications

View Applications

### Connections

Create Connection

View Connections

Sign Off



Fields marked with an asterisk (\*) are required.

Previous

Update and Previous

### Basic

Label\* Associated Insert Title Assoc

Operation Type\* Association 1-1 Description Assoc

Operation Code

INSERT INTO PJDATA.EMPLOYEE (WORKDEPT,EMPNO,  
FIRSTNAME,MIDINIT,LASTNAME,PHONENO  
,HIREDATE, JOB,EDLEVEL,SEX,BIRTHDATE,SALARY,BONUS,  
COMM) VALUES (??DEPTNO,?,?,?,?,?,?,?,?,?,?)

Instructions

### Display

Allow Details ☒ Display Group Defa

Allow Inserts ☒ Display Order 0

Allow Updates ☒ Display Columns

Allow Deletes ☐

DisplayColumns{ results;; details;; }

DetailDisplay{

buttonJustify;; colCnt;; copyURI;;

delete;; editURI;; insert;;

insertAndCopy;; insertAndNew;;

insertURI;;

labelJustify;; maxInputWidth;;

maxInputWidthSum;;

The operation code used for the Insert:

```
INSERT INTO PJDATA.EMPLOYEE (WORKDEPT, EMPNO, FIRSTNME, MIDINIT, LASTNAME,  
PHONENO, HIREDATE, JOB, EDLEVEL, SEX, BIRTHDATE, SALARY, BONUS, COMM) VALUES  
(??DEPTNO, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

Notice the SQL code is similar to the normal INSERT SQL statement. In this example, PJDATA.EMPLOYEE is the table into which data is inserted. The parentheses hold all the fields which are going to have information inserted. The VALUES clause tells where the values for this row, with columns specified, will come from. In this example, the WORKDEPT field, in EMPLOYEE, is being linked with DEPTNO field, which is from the DEPARTMENT table. The linking is done with the double question marks (??) and the field where the associated data is being retrieved from. The other single question marks in the parenthesis will take user input for the new row.

After creating the Associated Insert, you will see it in your list of operations but will not see it in your application. This is because your operation cannot be directly run. It can only be initiated once an associated row is available. Now you have to assign your association to a specific Field in the associated table (in our example, this is the DEPTNO field). To do this, run an operation that displays the table you are using for your association. In the example above, we are using the DEPARTMENT table. So we will run the operation to display the DEPARTMENT table. Then we will edit the field descriptor of the field in the table to be associated with the insert operation. In our example, we will edit the field descriptor of the DEPTNO field in the DEPARTMENT table.

Departments				
	▲ Department # ▼	▲ Department Name ▼	▲ Manager # ▼	▲ Administrator Dept ▼
	A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
	B01	PLANNING	000020	A00
	C01	INFORMATION CENTER	000030	A00
	D01	DEVELOPMENT CENTER		A00
	D11	MANUFACTURING SYSTEMS	000060	D01
	D21	ADMINISTRATION SYSTEMS	000070	D01
	E01	SUPPORT SERVICES	000050	A00
	E11	OPERATIONS	000090	E01
	E21	SOFTWARE SUPPORT	000100	E01
	F22	BRANCH OFFICE F2		E01
	G22	BRANCH OFFICE G2		E01
	H22	BRANCH OFFICE H2		E01
	I22	BRANCH OFFICE I2		E01
	J22	BRANCH OFFICE J2		E01

Insert

Once you have a screen similar to the one above, (but without the hyperlinks), you can connect the association you previously created. To do this click the gear symbol next to the field you want to link to your insert operation. In the Field Descriptor Screen scroll down to the Advanced Setting area (shown below) and set the Association Operation to your previously created Operation.

### Advanced Settings

Field Class:	<input checked="" type="radio"/> Enter Class Name: <input type="text"/>	<input type="radio"/> Select Existing: <input type="text" value="Address 1"/>
Field Descriptor Type *	<input type="text" value="Default"/>	Formatter Class: <input type="text" value="(None)"/>
Concurrency *	<input type="text" value="Concurrent Updates and Deletes Allowed"/>	Getter Method: <input type="text"/>
Remarks:	<input type="text"/>	Setter Method: <input type="text"/>
Association Operation:	<input type="text" value="WorkDeptAssoc"/>	XML Tag: <input type="text"/>

Once you have saved your changes to the field descriptor, the association is complete. Now, whenever the DEPTNO field of the Department table is displayed, it will have a hyperlink to insert into the employees table with the selected department number and asking the user for the other values.





Default

Employees

Departments



Fields marked with an asterisk (\*) are required.

Insert

Ca

EMPNO\*



FIRSTNAME\*

MIDINIT\*



LASTNAME\*

WORKDEPT



PHONENO

HIREDATE



JOB

EDLEVEL\*



SEX

BIRTHDATE



SALARY

BONUS



COMM

Insert

Ca

## Associated Updates

[EE] Creating an associated update is very similar to creating an associated insert, but instead of inserting a row with values from an associated row, it updates a row with values from an associated row.

Creating an Associated Update is very similar to doing basic 1- 1 or 1-many associations (described in at the beginning of this section). To create an association operation, begin by creating a new operation as described in the Operations chapter. Next, change the Operation Type from SQL to either the 1-1 Association or 1-Many Association Type. Then you need to set its operation code. After the operation code has been set, you need to modify a Field's Field Descriptor to set the association, so when field is generated it will have a link to update in its associated row or rows. This process is described below:

### SQL Associated Update Example

The operation code needed for an SQL Associated Update is very similar to the SQL statement for a normal update (described in the Update chapter), with some changes for the association.

Here is an example for a 1-1 Associated Update: after selecting the Create Operation from the TOC, select 1-1 for Association Type. The Operation Code is the SQL statement for updating the database. In our example we will be updating the SALARY field in the EMPLOYEE table by adding 1000 to the original value, linking from the Department Table. (We want to update all salaries for a single department only.) The code shown below will link the Department table to the Employee table allowing you to update entries in the Employee table with the similar fields WORKDEPT and DEPTNO:

The operation code used for the Update:

```
PDATE PJDATA.EMPLOYEE SET SALARY = SALARY + 1000 WHERE WORKDEPT = ??DEPTNO
```

Notice the SQL code is similar to UPDATE SQL statement described in the Update chapter. In this example, the PJDATA.EMPLOYEE table is being updated. The Set clause sets the salary equal to the current salary plus 1000. The WHERE clause, shows the which field the association is being updated from, in this example the WORKDEPT field in the EMPLOYEE is being linked with DEPTNO field which is the DEPARTMENT table. The linking is done with double question marks (??) and the name of the field from the associated row which is used in the update. After creating the associated update you will see it in your list of operations but will not see it your application. (This is because the associated update cannot be directly run, it must be invoked after the associated row has been retrieved.)

Now you have to assign your association to a specific Field - in our example this is the DEPTNO field. To do this run an operation that displays the associated table (in our case the DEPARTMENT table, shown below).

Departments				
	▲ Department # ▼	▲ Department Name ▼	▲ Manager # ▼	▲ Administrator Dept ▼
	A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
	B01	PLANNING	000020	A00
	C01	INFORMATION CENTER	000030	A00
	D01	DEVELOPMENT CENTER		A00
	D11	MANUFACTURING SYSTEMS	000060	D01
	D21	ADMINISTRATION SYSTEMS	000070	D01
	E01	SUPPORT SERVICES	000050	A00
	E11	OPERATIONS	000090	E01
	E21	SOFTWARE SUPPORT	000100	E01
	F22	BRANCH OFFICE F2		E01
	G22	BRANCH OFFICE G2		E01
	H22	BRANCH OFFICE H2		E01
	I22	BRANCH OFFICE I2		E01
	J22	BRANCH OFFICE J2		E01

Insert

Once you have a screen similar to the one above (without the hyperlinks) you can set up the association you previously created. To do this click the gear symbol next to the field you want along with your association. In the Field Descriptor Screen scroll down to the Advanced Settings area (shown below) and set the Association Operation to your previously created Operation.

Once you have saved your changes to the field descriptor, the association is complete. Now, whenever the DEPTNO field of the Department table is displayed, it will have a hyperlink to increment by 1000 the SALARY field of the associated rows in the EMPLOYEES table

## Associated Deletes

[EE] Associated deletes allow you to delete one or more rows based on the values contained in an associated row.

Creating an Associated Update is very similar to creating basic 1- 1 or 1-many associations (described in the beginning of this section). To create an operation for doing associated deletes, you first create a new operation and set its Operation Type from SQL to either 1- 1 Association or 1-Many Association Type. Then you set its operation code to do the actual delete; finally you attach the operation to an associated field by editing that field's field descriptor. This process is described in detail below:

## SQL Associated Delete Example

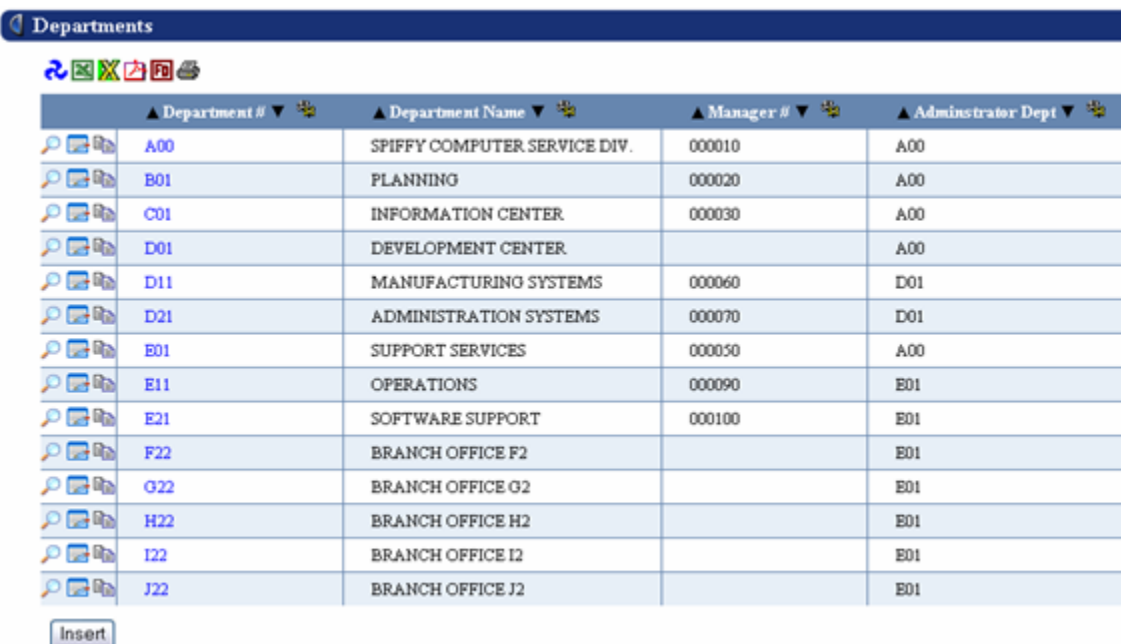
The operation code needed for the SQL Associated Delete is very similar to a normal SQL DELETE statement, with some changes for the Association. Our example will deal with a 1-Many Associated delete. After selecting the Create Operation from the TOC, choose 1-Many for Association Type and set the Operation Code to the SQL for performing the delete. The code shown below will link the DEPARTMENT table to the EMPLOYEE table allowing you to delete entries in the EMPLOYEE table whose WORKDEPT field matches the DEPTNO field of a row in the DEPARTMENT table:

The operation code used for the Associated Delete:

```
DELETE FROM PJDATA.EMPLOYEE WHERE WORKDEPT = ??DEPTNO
```

Notice the SQL code is similar to DELETE SQL statement described in the SQL Delete chapter. The WHERE clause links WORKDEPT field in the EMPLOYEE table to the DEPTNO field in the DEPARTMENT table. After creating the Associated Delete you will see it in your list of operations but will not see it your application. (This is because the operation can only be run after an associate row has been displayed.) The next step is to assign your association to a specific field in the associated table (DEPTNO). To do this run an operation that displays the associated table:

**Departments**



▲ Department # ▼	▲ Department Name ▼	▲ Manager # ▼	▲ Administrator Dept ▼
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
B01	PLANNING	000020	A00
C01	INFORMATION CENTER	000030	A00
D01	DEVELOPMENT CENTER		A00
D11	MANUFACTURING SYSTEMS	000060	D01
D21	ADMINISTRATION SYSTEMS	000070	D01
E01	SUPPORT SERVICES	000050	A00
E11	OPERATIONS	000090	E01
E21	SOFTWARE SUPPORT	000100	E01
F22	BRANCH OFFICE F2		E01
G22	BRANCH OFFICE G2		E01
H22	BRANCH OFFICE H2		E01
I22	BRANCH OFFICE I2		E01
J22	BRANCH OFFICE J2		E01

Insert

Once you have a screen similar to the one above (without the hyperlinks) you can set up the association you previously created. To do this click the gear symbol next to the field you want along with your association. In the Field Descriptor Screen scroll down to the Advanced Settings area (shown below) and set the Association Operation to your previously created Operation.

Once you have saved your changes to the field descriptor, the association is complete. Now, whenever the DEPTNO field of the Department table is displayed, it will have a hyperlink to increment by 1000 the SALARY field of the associated rows in the EMPLOYEES table.

## Join Associations

[EE] One widely used feature of SQL lets you combine, or "join" data from two tables into a single result table. If your data is on two separate systems however, you cannot use regular SQL to join it. Using associated joins, WOW gives you the ability to join data from two separate systems.

As an example, say we have a table (CUSTOMER) on one system with columns ID, NAME, and BALANCE; and another table (CUSTINFO) on a second system with columns ID and COLOR; and we want to join the two table together on the ID column, letting the user view a customer's name, ID, balance, and favorite color all in a single table. (For our example, we will assume that field descriptors for both tables have already been created, as described in the previous chapter, and that connections for both systems have been created.) The first step is to create the "base" query. This is a normal SQL Operation, selecting the rows of

interest from a single table:

Basic		
Label*	All Customers	Title
Operation Type*	SQL	Description
<pre>SELECT * FROM pjuser60.customer</pre>		






























In our example, we are selecting all the rows, but you can use any type of WHERE clause you wanted with this query.

The next step is to create the "join" query – which should select all the rows from the second table. Do not specify a WHERE clause in the join query. This operation's type must Associated Join:

Basic		
Label*	Customer-Color Join	Title
Operation Type*	Associated Join	Description
<pre>SELECT * FROM pjuser60.custinfo</pre>		

Note that you will want to specify a different connection alias for the join operation than you did for the base operation since they are on two different systems.

Next, start the application and run the base operation (only data from one table should be retrieved):

	▲ ID ▼ 	▲ Name ▼ 	▲ Balance ▼ 
 	1	Justin	4900
 	55	Ky	805
 	5	Sam	66
 	30	Red	150
 	40	Mikey	120
 	11	Rima	301
 	12	Jack	139
 	13	Terry	11
 	14	Don	34
 	15	Pico	110
 	16	Spike	477
 	17	Dennis	510
 	18	Velma	217

Click the gear icon to edit the FD of the column you want to join the two tables on. This

column must be common to both tables. In our example, this is the ID column. In the Field Descriptor Manager window, location the field descriptor's association operation, and set it to the Associated Join Operation we created earlier.

The screenshot shows the 'Advanced Settings' window for a field descriptor. The 'Association Operation' dropdown is highlighted with a red circle and is set to 'Customer-Color Join'. Other settings include 'Field Class' set to 'Enter Class Name', 'Field Descriptor Type' set to 'Default', 'Concurrency' set to 'Concurrent Updates and Deletes Allowed', 'Formatter Class' set to '-- None --', 'Getter Method' and 'Setter Method' are empty, 'XML Tag' is empty, and 'Notify Status Change' is set to 'No'.

Now when the base operation is run again, the results will a join between the two tables on different systems:

▲ ID ▼	▲ Name ▼	▲ Balance ▼	▲ Color ▼
1	Justin	4900	
55	Ky	805	
5	Sam	66	
30	Red	150	Puce
40	Mikey	120	Lemon
11	Rima	301	Orange
12	Jack	139	Brown
13	Terry	11	Purple
14	Don	34	Pink
15	Pico	110	Green
16	Spike	477	Copper
17	Dennis	510	Crimson
18	Vehma	217	Burgandy

## Possible Values

Possible Values are a crucial part of WOW. When a field has possible values, the application will display a drop down menu with the values that are possible for the field. This is important because it allows the user to pick a specific value instead of typing in a value that may or may not be valid. For example, let's say we want to create possible values so that when a user searches for an employee by department number, they can pick the department number from a drop down list of all department numbers.



To create Possible Values with WOW, you first need to create field descriptors for the table with which the possible values will be associated. In our example, this is the EMPLOYEE table. Once your table has field descriptors, the next step is to add a possible values operation to your application. To create a possible values operation, click on the "Create Operation" link that is visible when viewing a list of your application's operations.

Below is an example of setting up a possible values operation:

Label*:	WorkDept Possible Values	Title:	Department ID Possible Values
Type*:	Possible Values	Description:	Used to get Possible Values for Dep
Operation Code:	SELECT DISTINCT DEPTNO FROM PJDATA.DEPARTMENT		

To setup a possible values SQL operation, the Operation Type must be set to Possible Values. The SQL command above will select all of the distinct DEPTNO fields from the DEPARTMENT table (these are the values that the user will be able to choose from). DISTINCT is used so there is only one instance of each DEPTNO value.

After you have created the Possible Values Operation, you need to associate it with a specific field. This will be very similar to setting up Associations as described in the Association section of this guide. To associate our PV operation with a specific field, we will first run a query on the EMPLOYEE table to display all of its rows. The result will look similar to the screenshot below:

▲ Last Name ▼ 	▲ WORKDEPT ▼ 	▲ Phone # ▼ 	▲ Hire Date ▼ 
TstLNM	A00	3978	1965-01-01
THOMPSON	B01	3476	1973-10-10
KWAN	C01	4738	1975-04-05
GEYER	E01	6789	1949-08-17
STERN	E21	6423	1973-09-14
PULASKI	D21	7831	1980-09-30
HENDERSON	E11	5498	1970-08-15
SPENSER	E21	0972	1980-06-19
LUCCHESI	A00	3490	1958-05-16
O'CONNELL	A00	2167	1963-12-05

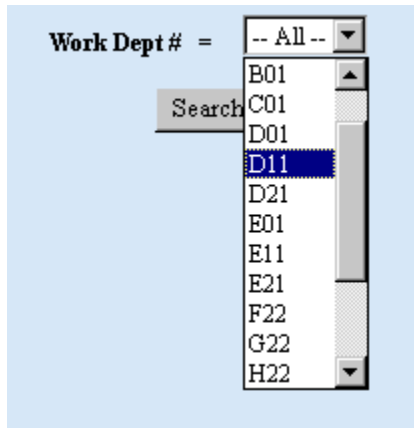
In the example above, we are setting up Possible Values for the WORKDEPT column. To setup the Possible Values, click on the 'gear' icon on the immediate right of the WORKDEPT column name. This will bring up the Field Descriptor Manager application in a new window. The only section we'll pay attention to will be the Possible Values Settings, which looks similar to the screenshot shown below:

Possible Value Settings			
Possible Values Key:	-- None --	Possible Values Operation:	Dept # PV
Possible Value Class:	<input checked="" type="radio"/> Enter Class Name: <input type="text"/>	<input type="radio"/> Select Existing Class:	<input type="text"/>
			*DISTINCT*

To complete your Possible Values, find the correct Possible Value that is listed under the Possible Values Operation (in the example, the "Dept # PV" was chosen). The Operations listed are all Possible Value Operations that have been created for your specific WOW application. After finding the corresponding Possible Value Operation, update the screen and your Possible Value will be setup and ready to use.

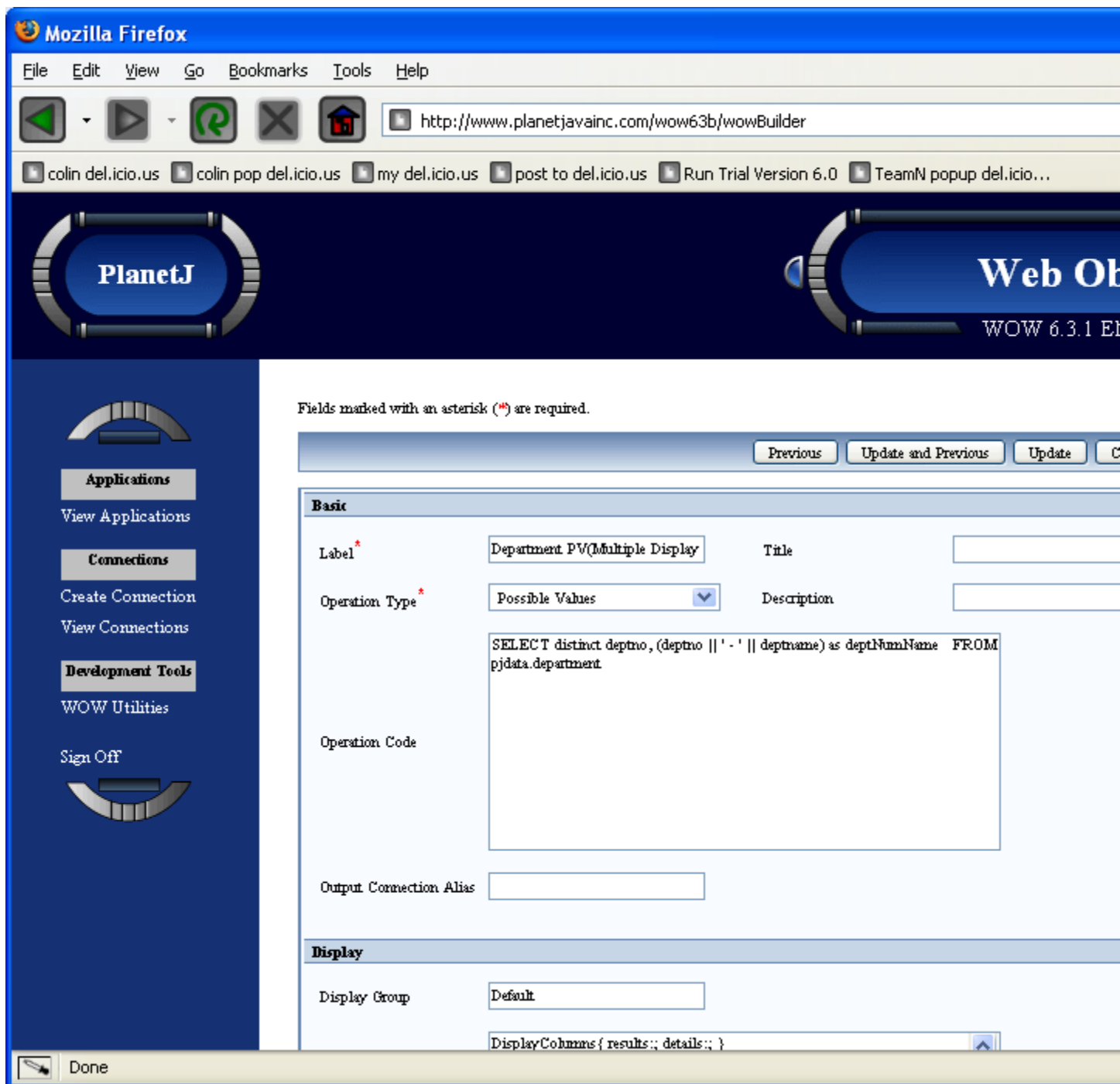
Now when we run an operation with this code: `SELECT * FROM PJDATA.EMPLOYEE WHERE WORKDEPT = ?`, we will get a drop down with all the possible department numbers available:





## Multiple Fields in Possible Values Drop Down

When creating a Possible Values operation, there are times where you may want the Possible Values Drop Down to include two or more fields to give the user more feedback and information. When creating a Possible Values operation, such as the one created, the first field selected is the value or field that is inserted into the database. The second field is the display value, which is what are going to change so that both department number and name are shown in the possible values drop down. In the possible value operation, we need to change the SQL code so that it adds the deptno field and the deptname field together as shown below:



Operation Code:

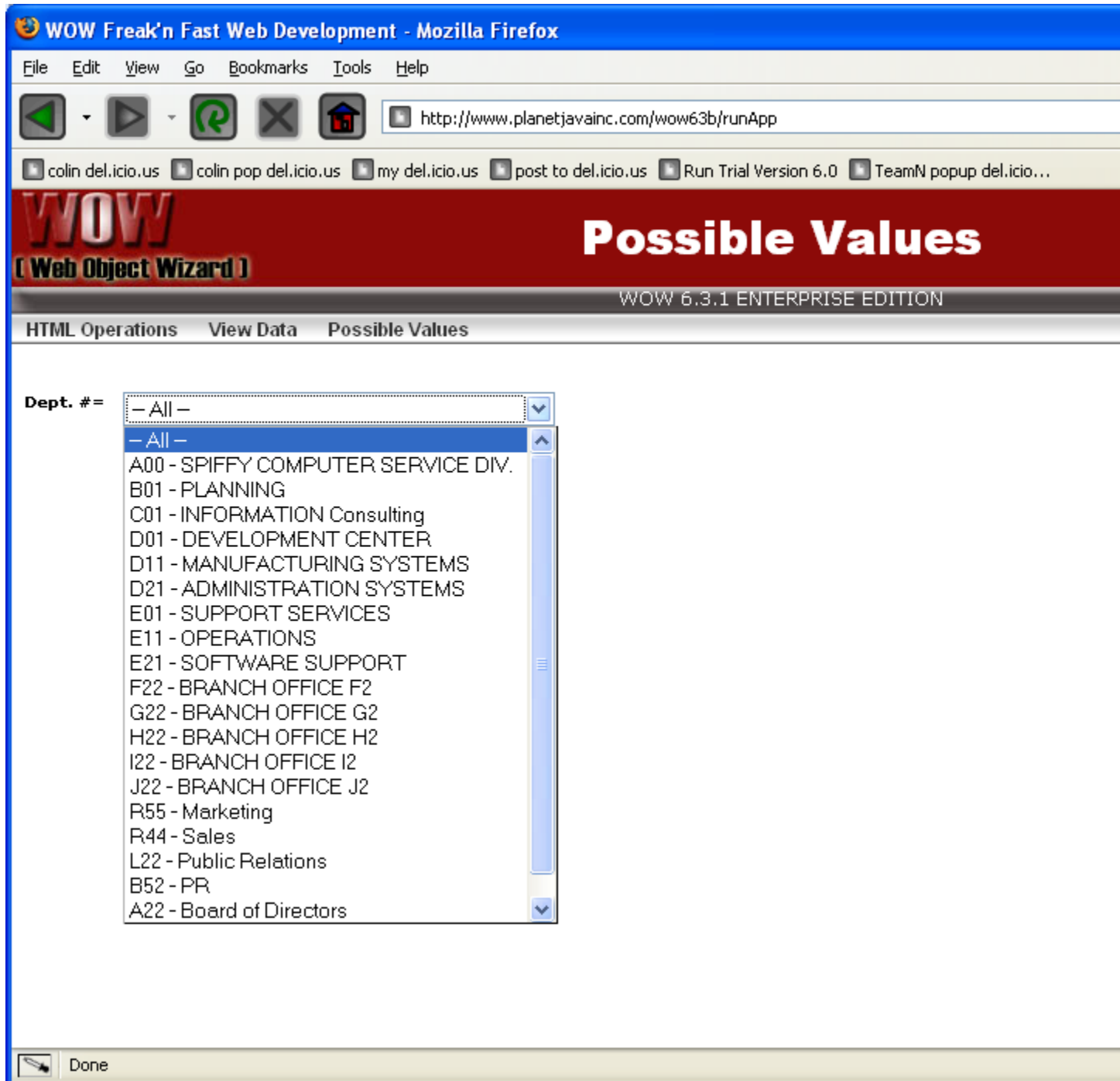
```
SELECT distinct deptno, (deptno || ' - ' || deptname) FROM pjdata.department
```

When accessing data from an iSeries, you should use the || operator to concatenate fields together for the display value. If you are using MySQL, then you need to use the CONCAT () function instead of the || command. In this case the operation code would look like this:

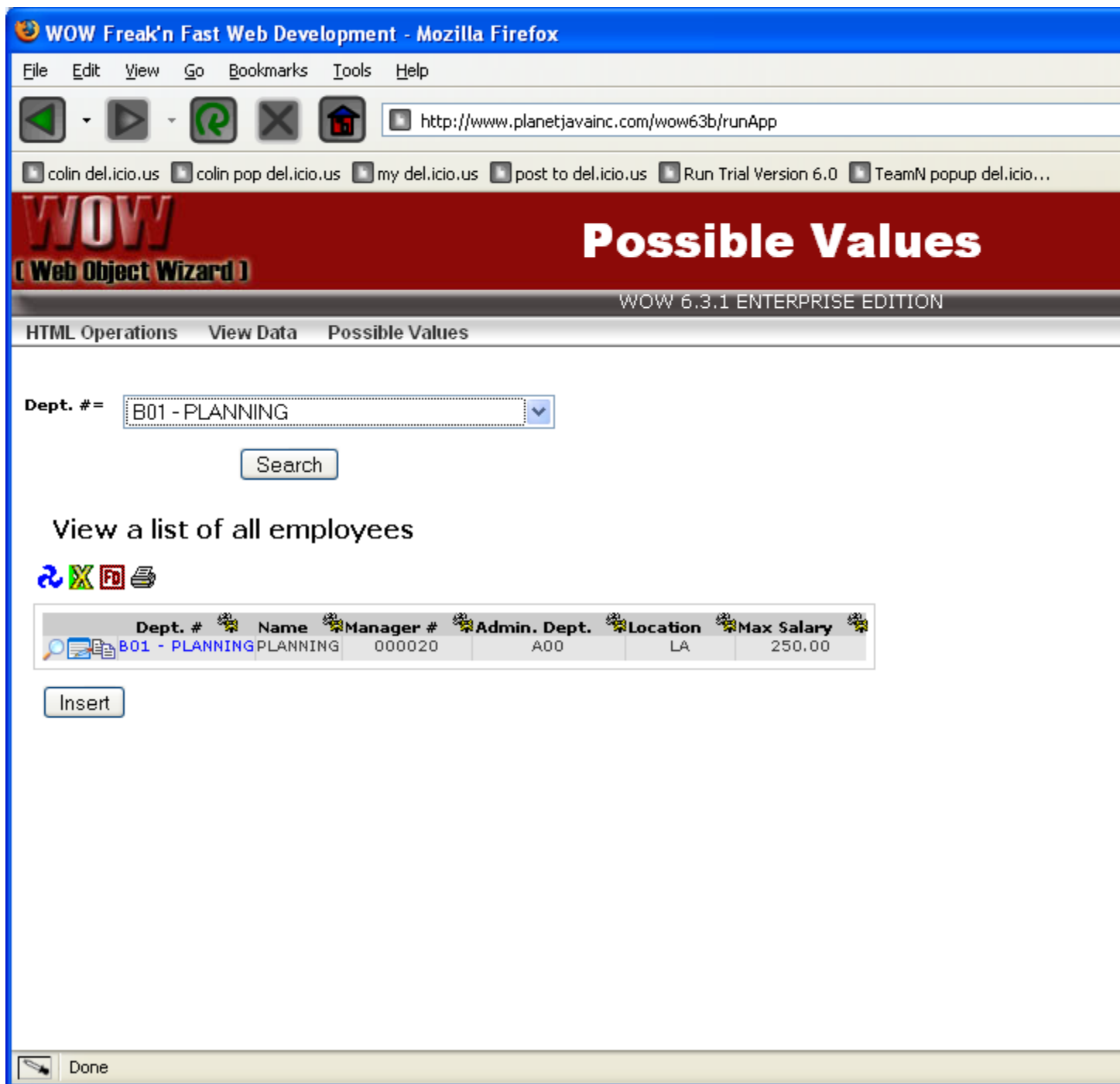
Operation Code:

```
SELECT DISTINCT deptno, CONCAT(deptno,CONCAT(' - ', deptname)) FROM  
pjdata.department
```

In the above example, the deptno is displayed with a dash and then the department name is shown:



When "B01 – Planning" is selected, the SQL statement that runs uses "B01" as the actual value:



## Possible Values and the – All – Value

In the above screen shot, notice that WOW has added in a special "– All –" value to the list of department numbers. The all option lets the user search for employees with any department number. However the "– All –" choice will only work correctly if your SQL has been coded properly to handle it.

The "– All –" value always corresponds to a NULL SQL value. This means if your SQL statement was `SELECT * FROM PJDATA.EMPLOYEE WHERE WORKDEPT = ?` and the user

selected the "- All -" value, no rows would be returned. This is obviously an incorrect result. The proper SQL in this case would be `SELECT * FROM PJDATA.EMPLOYEE WHERE WORKDEPT = COALESCE(CAST(? AS CHAR(3)),WORKDEPT)`. This SQL is written so that if the value supplied by the user is NULL, then all rows of the EMPLOYEE table are returned, regardless of the department number.

### Customizing the - All - Item

Although WOW puts the "- All -" item into search parameters by default, you can change this text to anything you like, or even remove it altogether. This text is controlled by the `dropDownItemDisplay` property of the `OperationLabels` property group (property groups were covered earlier in this chapter). If you wanted the text to say "- Choose -" instead of "- All -" you would insert the following text into the properties field of your operation:

```
OperationLabels { dropDownItemDisplay: - Choose -; }
```

If you like, you can also instruct WOW to eliminate this extra item altogether. This is done by specifying NULL as the value of the `dropDownItemDisplay` property:

```
OperationLabels { dropDownItemDisplay: NULL; }
```

**NOTE:** The `OperationLabels` property group is specified with the regular current operation, NOT with the possible value operation.

### Further Customizing the - All - Item

Whether you change how the "- All -" item is displayed or not, by default the value that is actually sent to WOW and placed in your query is the special null value. If you like, you can choose to have a different value placed into your query when this item is selected. Just specify the value you want using the `dropDownItemValue` property:

```
OperationLabels { dropDownItemDisplay: - Choose -; dropDownItemValue:
Nothing; }
```

The above example would add an item to the possible values drop down with a display text of "- Choose -". When this item is selected, the value "Nothing" would be sent to WOW.

**NOTE:** You cannot specify a value for the `dropDownItemValue` property unless you also specify a value for the `dropDownItemDisplay` property.

### Removing the - All - Item in a Search

To remove the "- All -" item from a search prompt, go to the FD of the relevant field and check the "Required on Search" box. This will remove the "- All -" option and force the user to select a value. This is particularly useful when you have two (or more) different drop down fields in one SQL operation and only want one field to have the "- All -" item. Rather than using the `OperationsLabels` feature which applies to all fields in the operation, you would use the "Required on Search" feature to selectively remove the "- All -" item.

PlanetJ

Field Description

TESTC

Fields marked with an asterisk (\*) are required

Previous Update and Previous Update Cancel Delete Up

Basic Settings

Field Name\*: LSTNAM External Name: Last Name

Required: ☒ Required On Search: ☒

Default Value: ☐ None ☐ Auto Update Value:

Table FDs

AutoTrim

Default Value

Display Properties

Shared FDs

Table FDs

All FDs

## Removing – Next – and – Previous – from Possible Value List

If you have a long list of choices resulting from your possible values (PV) operation, you may see the choices "– Next --" and "– Previous --" (added automatically by WOW). If you want to eliminate those choices, increase the Row Count value in the Advanced section of your possible values operation. For example, if your Row Count is set to 25 and the total number of choices returned from your PV operation is 35, increase the Row Count to a value larger than 35.

## PV Multiple Selects

There may be times when an end user wants to select multiple items from a drop down pick list. This can easily be accomplished using the SQL IN function. The SQL IN function helps reduce the need to use multiple OR conditions.

The syntax for the IN function is:

This SQL statement will return the records where column1 is value1, value2..., or value\_n.

The IN function can be used in any valid SQL statement - select, insert, update, or delete. To produce a drop down (PV list), use the WOW "?" function in combination with the IN function to produce multiple selects available via hitting CTRL key to multi select.

Example: SELECT \* FROM PJDATA.EMPLOYEE WHERE WORKDEPT IN ?

This will select employees from multiple departments. As shown in the screens below, you must first create the SQL operation:

**SELECT columns**  
**FROM tables**

**WHERE column1 in (value1, value2, .... value\_n);**

The screenshot displays the 'Web Object Wizard' interface for 'PLANETJ CORPORATION'. The top navigation bar includes 'Project Schema: WOWSAMP', 'Dev Schema: wowsamp', and 'Application: Multi-Select PV Example'. The main title is 'Web Object Wizard' with a version note 'WOW Version: 7.0.00 License: ENTERPRISE'. The left sidebar contains a 'First Time User?' section with a video link and a 'Web Application Creation Steps' section with three numbered steps: 1. Setup Connection(s), 2. Setup Application(s), and 3. Setup Operation(s). The main area is titled 'Basic' and contains a form for configuring a web object. The 'Operation Type' is set to 'SQL'. The 'Operation Code' field contains the SQL query: 'SELECT \* FROM PJDATA.EMPLOYEE WHERE WORKDEPT IN ?'. The 'Description' field contains the text: 'View a sample database of employees.' The 'Title' field contains the text: 'Select Multiple Values'. The 'Label' field contains the text: 'Select Multiple Values'. The 'Instructions' field contains the text: 'Select employees from multiple departments'. The 'Database Explorer' button is visible on the right side of the SQL query field.

Project Schema: WOWSAMP Dev Schema: wowsamp PLANETJ CORPORATION

Application: Multi-Select PV Example

Web Object Wizard  
WOW Version: 7.0.00 License: ENTERPRISE

Hide Side Steps Development Tools Services Products Log Out Help

First Time User?  
Check out this quickvideo to get started building a simple application!

Web Application Creation Steps:

- 1 Setup Connection(s)
- 2 Setup Application(s)
- 3 Setup Operation(s)

Fields marked with an asterisk ( \*) are required.

References Edit FDs Update Operation Cancel Delete Operation Update and Next Next

**Basic**

Label ? Select Multiple Values Title ? Select Multiple Values

Operation Type ? SQL Description ? View a sample database of employees.

Operation Code ?

SELECT \* FROM PJDATA.EMPLOYEE WHERE WORKDEPT IN ?


Database Explorer

Instructions ?

Select employees from multiple departments

Then, you will need to also create a PV operation to bring the various work departments into the drop down like this:

References
Edit FDs
Previous
Update and Previous
Update Operation
Cancel
Delete Operation



Web Application Creation Steps:

1

Setup Connection(s)

2

Setup Application(s)

3

Setup Operation(s)

4

Run!

Contact PlanetJ

support@planetjavainc.com

Basic

Label ?

PV: FIELD->PJDATA.EMPLOYEE.WORKDEPT

Title ?

PV Operation

Operation Type ?

Possible Values

Description ?

SELECT DISTINCT WORKDEPT,WORKDEPT AS DisplayValue FROM PJDATA.EMPLOYEE WHERE WORKDEPT = WORKDEPT

Operation Code ?

Database Explorer

Output Connection Alias ?

Display

Allow Details ?

☐

Display Rule ?

-- None --

Allow Inserts ?

☐

Display Location ?

-- None --

Allow Updates ?

☐

Display Group ?


PV: FIELD->PJDATA.EMPLOYEE

Display Order ?

11700

DisplayColumns{ results:: details:: }

Now, when running the application, a user can click on the drop down and hold the CTRL key down to select multiple departments:



Multi-Select PV Example

POWERED BY  
WOW

Employees in Departments

Select employees from multiple departments

Work Dept


IN

-- All --

SPFFY COMPUTER SERVICE DIV 2


QCT

SPFFY COMPUTER SERVICE DIV



Returned results are shown here:





POWERED BY  
WOW

---

**Employees In Departments**

Select employees from multiple departments

Work Dept IN 

-- All --  
 SPIFFY COMPUTER SERVICE DIV 2  
 QCT  
 SPIFFY COMPUTER SERVICE DIV

Select Multiple Values

▲ EMPNO ▼	▲ First Name ▼	▲ MIDINIT ▼	▲ Last Name ▼	▲ Work Dept ▼	▲ Phone ▼	▲ Hire Date ▼	▲ Job ▼	▲ Ed Level ▼	▲ Gender ▼	▲ Birth ▼
1004	Huong	f	Grimes	SPIFFY COMPUTER SERVICE DIV 2	858-241-4156	01/04/2008	CLERK	15	F	01/16/198
102579	Lewis	S	Libman	SPIFFY COMPUTER SERVICE DIV 2	2118	12/06/1999	MIC	15	M	12/07/196
123783	Jose	Q	Smith	SPIFFY COMPUTER SERVICE DIV 2	858-213-4213	03/03/2007	operator	12	M	01/14/197
100038	John	J	Smith	SPIFFY COMPUTER SERVICE DIV 2	253-555-1234	08/21/2007	Designer	12	M	01/20/197
87965	Angela	A	Jackson	SPIFFY COMPUTER SERVICE DIV 2	0672	01/01/2007	CLERK	0	M	05/11/200
000322	Kate	T	C-Note	QCT	858-327-5624	01/02/2008	MIC	14	F	01/16/198
0345	Stevie	K	Jones	QCT	858-315-9078	04/09/2007	clerk	12	M	04/11/198
100044	Hellen		Burger	QCT	858-2421456	01/01/2008	tech	12	F	01/11/198
100052	mot		ten	QCT	858-313-2561	10/03/2007	MANAGER	18	F	01/06/198

You can see the two different selected work departments rendered. If you want to change the size of the Possible Values box that is rendered, you can do that by changing the Display Height in the Field Descriptor here:

Usagelds   Shared FDs   Table FDs   Home   Search By   Quick Edit   Log Out

Fields marked with an asterisk ( ) are required.

**Basic Settings**

Field Name ?

Required ? ☐

Default Value ?

☐ -- None --  
☐

External Name ?

Required On Search ? ☐

Auto Update Value ?

**Display Settings**

Field Set ?

☒ Basic Info  
☐

Display Rule ?

☐ Always  
☐

Help Text ?

Display Width ?

Display Order ?

Display Component ?

☐ (Default)  
☐

Style Class ?

☒ -- None --  
☐

Display Height ?

**Possible Value Settings**

Possible Values Key ?

☐ -- None --  
☐

Possible Values Operation ?

☐ Work Dept Possible Values  
☐

Now, the user has a larger drop down box to select from:

**Employees in Departments**


Select employees from multiple departments

Work Dept

IN

-- All --

SPIFFY COMPUTER SERVICE DIV 2  
QCT  
SPIFFY COMPUTER SERVICE DIV  
Board of Directors  
PR1  
INFORMATION Consulting  
DEVELOPMENT CENTER  
MANUFACTURING SYSTEMS  
ADMINISTRATION SYSTEMS  
SUPPORT SERVICES  
OPERATIONS  
SOFTWARE SUPPORT  
BRANCH OFFICE F2  
BRANCH OFFICE G2  
Education  
BRANCH OFFICE H2  
BRANCH OFFICE I2  
BRANCH OFFICE J2  
WOW Support



## Possible Values Paging (Next/Previous)

If you have a long list of choices resulting from your possible values (PV) operation, you may see the choices "— Next —" and "— Previous —" (added automatically by WOW). You can click on the "-Next-" to see the next result of Possible Values. If you want to eliminate those choices or increase the # returned for each page of PV, increase the Row Count value in the Advanced section of your possible values operation. For example, if your Row Count is set to 25 and the total number of choices returned from your PV operation is 35, increase the Row Count to a value larger than 35. A better option for a large # of Possible Values is the Possible Values Search operation.

## Possible Values Grouping [Minimum Version: WOW 6.6 beta]

Recently, functionality was added to allow developers to group their Possible Values inside the drop-down menu itself. For example, a drop down may contains states and provinces that you want grouped by country to allow a more user friendly approach to find states/provinces directly for that country. Technically, behind the scenes, this new functionality uses the standard HTML **optgroup** element and then a padding-left CSS style added to our default theme to indent options when inside an option group. Here is an example of a possible value with the new grouping feature, i

n this example we're looking at subsystems which are grouped by system area:

Fields marked with an asterisk (\*) are required.

The screenshot shows a configuration window with a 'Subsystem' dropdown menu. The menu is open, displaying a list of options: -- Choose --, -- Choose --, asic, apps, build, core, api, apps, boot, bsp, buses, cs, dal, debugtools, hal, hsusb, hwengines, iodevices, kernel, mproc, and pkg. The 'asic' and 'core' options are highlighted with red boxes. The window also features 'Insert' and 'Cancel' buttons at the top right, and a red 'OP' button on the right side of the dropdown menu.

In the above screen shot, the following property group is set on the Possible Values operation: ***PossibleValue { optgroup:AreaName; }***

To use, add the PossibleValues{ } property group to the Possible Values operation and set optgroup to the name of the field that contains the optional group value. PossibleValues { optgroup:<field name>; }

**NOTE:** During generation, a new option group is started when the next option group field's value changes.

**NOTE:** Developers should also sort their Possible Values Operation's SQL first by the optgroup field, otherwise options in the same group could/will not actually be grouped together.

## Possible Value Keys

WOW also comes with several predefined Possible Values. There are Possible Values for US States, days of the week, and several other common scenarios. To select one of these predefined possible values for a field, use the Possible Values Key drop down in the field descriptor.

It is also possible to create your own possible values keys and have them appear alongside the predefined keys in the drop down. The process of creating your own possible values keys is described in the Possible Values section of the WOW Utilities chapter.

## Possible Values Selector

This operation is very powerful but requires a few steps in order to utilize. This operation is capable of setting several field values in a row based on the selection of a possible value. Consider an "order"; an order normally requires many fields to be set in the order header record. These fields may include the customer number, customer name, shipping address, etc. When you select the customer for an order, you want to "select" other fields to be copied into the row.

The SQL specified in this operation retrieves the Possible Values for the field and displays them like a normal Possible Values operation. The difference for the Possible Values Selector is when the user selects a value from the Possible Values drop-down, a call is made to the server which calls the method "handlePossibleValueOperation" on the field associated with this operation. The default behavior is to copy the values of the operation's SQL query via common usage id into the source row. The user may also specify to copy via common field name. This setting is determined by the value of the "copyRule" property of the Display Groupings Property Group. The valid values for this property are `usageid` and `fieldNames`.

For the following example, consider the following 2 tables:

### Customer file

CustomerId	LastName	FirstName	CustZip
1	Jones	Paul	92029
2	Lawson	Fred	57401

### Order File

OrderId	OrderCustId	OrdFirstName	OrdCustZip	OrdLastName
1	1	Paul	92029	Jones
2	2	Fred	57401	Lawson

### Follow these steps to utilize this operation:

1. Typically you will have a normal SQL edit or insert operation. Ex: `INSERT INTO mylibrary.myOrderFile.`
2. Create an operation of type `PossibleValueSelector` with the following SQL: `SELECT customerId AS OrderCustId, lastName || firstName as FullName, lastName`

```
as OrdLastName, firstName as OrdFirstName, custZip as OrdCustZip FROM  
myLib.customerFile
```

- a. Notice the use of the "as" feature to map customer field names to their corresponding order field names. WOW can now copy the fields from the Customer file into their matching fields in the Order file.
  - b. Also, it is very important to note that the first two columns selected behave just like a normal PV operation. The first column is the internal value while the second column is its corresponding external value. In the case above, the customer ID (internal value) is masked by the customer's full name (external value). This only applies to the first two columns. the remaining selected columns only contain internal values.
  - c. In the properties of this operation, you must tell WOW to map using the field names. You can do this by specifying the following property group:  
`OperationSettings{copyRule:fieldNames;}`
3. Now open the field descriptor on the "OrderCustId" field and set the possible value operation field to the operation created in step 2. Also, set the field descriptor's "Status Change" to yes, which will force a screen refresh when a new value is selected. At this time, WOW will attempt to copy the fields from the possible value row to your current row.
4. As an alternative, you can also set the usage ID values in both the customer and order file and WOW will copy the fields that have matching usage ID values. In this scenario you would specify the following property group:  
`OperationSettings{copyRule:usageid;}`.

## Possible Values Search

Possible Values Search operation allows you to create any operation with or without search parameters to find a particular possible value to use for a field. WOW opens the Possible Values Search operation in a separate pop-up window, runs it after you have specified parameters and then the user selects the correct value. For example you may have a possible values operation that returns two thousand options for the user to select; this can cause problems because of the extremely large size of the drop-down. In this case you would want the user to be able to search or query down to a more manageable list of options and then select the correct value.

### Steps to Utilize Possible Values Search Operation:

1. Create Possible Values Search Operation  
Create new operation and set the operation type to Possible Values Search. In operation code create a standard SQL select statement; just make sure you specify the Possible Value field first. Note: Job is the first field in the select because you will set this PV Search operation to the Job field. The first field returned should always be the field with a value that you want to use as the Possible Value. After specifying the other fields plus any search conditions that you want to present the user, then click the create operation button.

### First Time User?

Check out this [quick video](#) to get started building a simple application!



### Web Application Creation Steps:

1

[Setup Connection\(s\)](#)

2

[Setup Application\(s\)](#)

3

[Setup Operation\(s\)](#)

4

[Run!](#)

### Contact PlanetJ

[support@planetjavainc.com](mailto:support@planetjavainc.com)

### Basic

Label\*

Job PV Search

Operation Type\*

Possible Values Search

```
SELECT job, empno, firstnme, las  
lastname like ?
```

Operation Code

Instructions

Output Connection Alias

### Display

Allow Details

☐

Allow Inserts

☐

Allow Updates

☐

2. Set Possible Search Operation to desired field's field descriptor

Run an operation that includes the field that you would like to set the PV Search operation and open that field's Field Descriptor. If no Field Descriptor exists then create Field Descriptors for the entire table. In this case we set the PV Search operation to the Job field in a sample employee table as shown below.





3. Update or Insert with new Possible Value Search

[EE] When you edit (update or insert) a record with the Job field WOW will generate a retrieve button next to the field. This will retrieve the user's desired possible value by running the Possible Value Search operation in a new pop-up window.



# SP2 Features App

Default

Previous

Update and Previous

Update

Cancel

Update and Next

<b>EMPNO</b>	<input type="text" value="000060"/>	<b>First Name</b>	<input type="text" value="Doddy"/>
<b>MIDINIT</b>	<input type="text" value="F"/>	<b>Last Name</b>	<input type="text" value="STERN"/>
<b>WORKDEPT</b>	<input type="text" value="R44"/>	<b>PHONENO</b>	<input type="text" value="6423"/>
<b>HIREDATE</b>	<input type="text" value="09/14/1973"/>	<b>JOB</b>	<input type="text" value="MANAGER"/> <input type="button" value="Retrieve"/>
<b>EDLEVEL</b>	<input type="text" value="25"/>	<b>Gender</b>	<input type="text" value="Male"/>
<b>BIRTHDATE</b>	<input type="text" value="12/07/1966"/>	<b>SALARY</b>	<input type="text" value="47650.00"/>
<b>BONUS</b>	<input type="text" value="100.00"/>	<b>COMM</b>	<input type="text" value="1680.00"/>
<b>PWD</b>	<input type="text" value="rty"/>	<b>USERID</b>	<input type="text" value="STERN"/>

IMAGE

images/pic\_10.jpg

Previous

Update and Previous

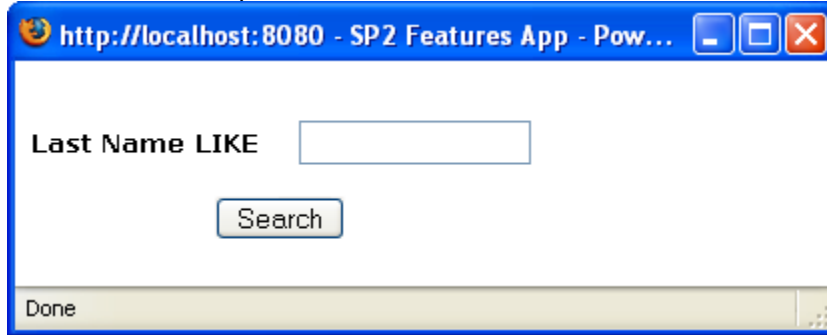
Update

Cancel

Update and Next

4. Retrieve Possible Values

The new pop-up window will run the Possible Values operation in this case searching last name. After entering in a search parameter value, WOW will bring back a resultset with all possible values.



5. Populate Field with valid Possible Value (PV)

The returned results have a populate button that when clicked will grab the first field's (column) value and set as the current field's PV. In this example the first field is Job and the first populate button is clicked fills the 'MANAGER' value into the field as seen below.

http://localhost:8080 - SP2 Features App - Powered by PlanetJ's WOW - Mozilla Firefox

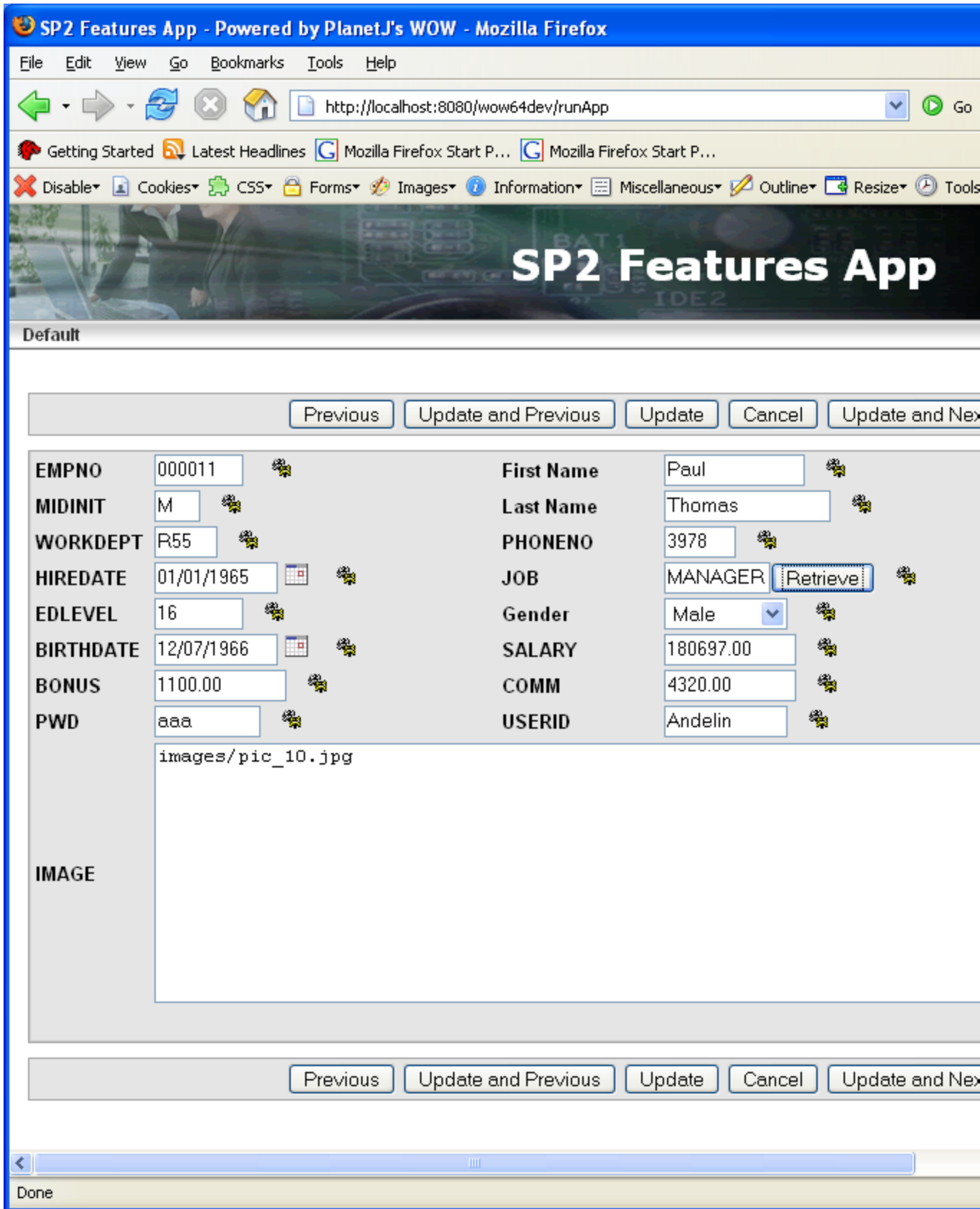
Last Name LIKE

### Job Possible Values

↻

	▲ JOB ▼	▲ EMPNO ▼	▲ First Name ▼	▲ Last Name ▼	▲ WORKDEPT ▼
<input type="button" value="Populate"/>	MANAGER	000060	Doddy	STERN	R44
<input type="button" value="Populate"/>	MANAGER	000100	Paul	SPENSER	F22
<input type="button" value="Populate"/>	CLERK	000250	DANIEL	SMITH	L22
<input type="button" value="Populate"/>	OPERATOR	000300	PHILIP	SMITH	E11
<input type="button" value="Populate"/>	OPERATOR	200280	EILEEN	SCHWARTZ	E21
<input type="button" value="Populate"/>	OPERATOR	200310	MICHELLE	SPRINGER	E21
<input type="button" value="Populate"/>	Maint	44873	John	Swift	O14
<input type="button" value="Populate"/>	Consulta	4721	Cynthia	Strauss	A22
<input type="button" value="Populate"/>	Maid	778392	Lydia	Swanson	O13
<input type="button" value="Populate"/>		987352	John	Sk	F22
<input type="button" value="Populate"/>	MANAGER	987368	Doddy	STERNS	R44

Done



Now that the value has been filled in proceed as normal with your update, insert or search. There is no limit to the # of fields that can have Possible Value Search in a particular operation.

### Using Possible Values Search to Populate Other Fields

Possible Values (PV) Search can be configured to populate other values/fields (similar to Possible Values Selector). The PossibleValues property group is used for this function and must be placed in the main (not the PV search) operation's properties. In this example, the PV Search operation should have in it's results the first 3 fields equal to (same field names, same order, similar data types)) the copyList fields(BasePath, BranchType, BranchName):

PossibleValues { fieldName:BasePath; copyList:BasePath,BranchType,BranchName; }

- *fieldName* - This property identifies which field in the main operation this configuration belongs to. It is needed in case your operation has more than 1 PV Search.
- *copyList* - A list of fields to copy.
- *copyRule* - fieldNames, usageId. Defaults to fieldNames. Tells how field values from the possible value row should be filled into/mapped to the actual row once a possible value is selected.

**NOTE:** If using field names (default) for *copyRule*, make sure the field names match between the PV Search operation and the main operation. Also, the *copyList* field names must be listed in the same order as they are listed in the PV Search operation's SQL.

**NOTE:** If the *copyList* Fields do not match with fields in the main operation, a JS error will occur and the Populate button will not function correctly.

## Auto Population of Fields

To demonstrate this operation, we will be using the same sample data in PJDATA: the EMPLOYEE and DEPARTMENT tables. We will be using the DEPTNO field of the DEPARTMENT table as the Target Field to match up with the WORKDEPT field of the EMPLOYEE table. First, you need to create an operation to display the DEPARTMENT Data, such as "SELECT \* FROM pjdata.department". Next, create a new operation and set the operation type to Auto Populate. An Auto Populate operation will show up in the list of operations for an application; however, it will not show up in the application itself. Create the SQL statement that matches up the field with the information to fill the new row.



### Applications

View Applications

### Connections

Create Connection

View Connections

Sign Off



Fields marked with an asterisk (\*) are required.

Previous

Update and Previous

#### Basic

Label\* Sample Auto Populate Title Samp

Operation Type\* Auto Populate Description View

Operation Code  
SELECT WORKDEPT,EMPNO FROM PJDATA.EMPLOYEE  
WHERE WORKDEPT = ??DEPTNO

Instructions

#### Display

Allow Details ☒ Display Group Defa

Allow Inserts ☒ Display Order 0

Allow Updates ☒ Display Columns

Allow Deletes ☐



```
SELECT WORKDEPT, EMPNO FROM PJDATA.EMPLOYEE WHERE WORKDEPT = ??DEPTNO
```

The SELECT command retrieves the fields WORKDEPT and EMPNO FROM the EMPLOYEE table. Next, the WHERE statement specifies which field the association is aligned with. In this example, the WORKDEPT field in the EMPLOYEE table is being linked with DEPTNO field in the DEPARTMENT table. The link between the two tables is established with the use of the double question mark (??). This special WOW Builder syntax tells WOW to take the value for DEPTNO from the current row, which is coming from the DEPARTMENT table. Now the field needs to be notified that it should use the Auto Populate operation. In the FD Manager, set the Possible Values Operation for the Field that you want to have the retrieve button generated next to. In this example, this would be the DEPTNO field.

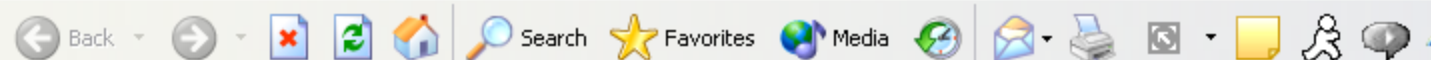
The key to an Auto Populate operation is that it fills in values for other fields in the same row as the field that the operation is associated with. This is accomplished by specifying a usage ID on the fields that need to be populated. If you are pulling information from a file that has different fields than the file that your detail Row came from, you must specify the FDs for the fields that return from the file.

Once you have created these FDs, assign the same usage ids to the fields so they will match up to the fields in the Detail Row (source Row). Certain usage ids are set aside for special designations such as Email Field (-40), Password Field (-80), and State Field (-120). Special system usage ids are always negative. You should pick an arbitrary positive # to start your usage ids from. For this example, the starting value of the usage ID is 5000.

Once again, you must assign a usage ID to each field that is returning from the query, and also assign that same usage ID to the respective FDs in the Detail Row. For instance, if the field name in the Detail Row was WORKNO and the field name in the file that you are retrieving it from was WORKDEPT, you must assign the Usage ID 5000 to both of those fields so that they will match up in the copy. The Usage ID is set in the Additional Settings group and the Auto Population operation is set to the possible values operation in the Possible Value Settings Group (shown below).

Field Descriptor Manager - SAMPLES60 - Microsoft Internet Explorer

File Edit View Favorites Tools Help



Address [http://www.planetjavainc.com:82/wow/runApp?ACTION\\_REQUESTED=FDMGR\\_ACTION\\_VIEW\\_FD&id=%2D%32&OWID=2307&PJ\\_SY](http://www.planetjavainc.com:82/wow/runApp?ACTION_REQUESTED=FDMGR_ACTION_VIEW_FD&id=%2D%32&OWID=2307&PJ_SY)

Search for  Search

Display Properties

Edit Field Remarks

Shared FDs

Table FDs

All FDs

UsageIds

Table FDs w/UsageId

Table FDs

All FDs w/UsageId



Required: ☒

Default Value:

Auto Update Value:

Display Settings

Field Set:

Display Order:

Display Rule: Always

Display Component: System

Help Text:

Style Class:

Display Width:

Display Height:

Possible Value Settings

Possible Values Key: -- None --

Possible Values Operation: Select

Possible Value Class: ☒ Enter Class Name:

☐ Select Existing Class

\*DISTINCT\*

Advanced Settings

Field Class: ☒ Enter Class Name:

☐ Select Existing:

Field Class:

Address 1

Field Descriptor Type: Default

Formatter Class:

Concurrency: Concurrent Updates and Deletes Allowed

Getter Method:

Remarks:

Setter Method:

Association Operation: -- None --

XML Tag:

Notify Status Change: No

Authorization Settings

Read Authorization Operation: 0

Edit Authorization Operation: 0

Additional Settings

Sortable: ☒

Auto Increment: ☐

Read Only: ☐

Currency: ☐

Id: 37060

Usage Id: 5000

Following these steps will generate the "Retrieve" button next to the DEPTNO field on the Insert Screen. Enter a department number into the DEPTNO field and press "Retrieve". The query specified in the Auto Populate operation will be executed. Any resulting fields that come back from the query that have corresponding usage IDs in the source Row will be filled in. For example, we used the EMPNO in the SQL statement and it can match up with the MGRNO. The EMPNO and MGRNO fields need to have the same usage ids to match up, we used 501. Now the manager number will fill with employee table data when the retrieve button is pressed with valid department number.

# Auto Populate Sample

Welcome



## Default

Employees

Departments



Fields marked with an asterisk (\*) are required.

Insert

DEPTNO \*

A00

Retrieve



DEPTNAME \*

MGRNO

000010



ADMRDEPT \*

LOCATION



Insert

## Execution Groups

An execution group is an operation that runs multiple operations simultaneously. Rather than having multiple operations running separately, they can be grouped together underneath one single parent operation.

### Create A Working Execution Group

To create an execution group, you first create a new or edit an existing operation. Change the **Operation Type** under **Basic** settings to **Execution Group** and create/update the operation.

This operation will be the parent operation; you must now create its child operations. To do so, create or edit another operation and, underneath the **Advanced** settings header, change the **Parent Operation** to the operation created in the previous step.

If you were to run the application, you would find the parent operation of the execution group (the first operation made in this tutorial) in the menu. If you open the parent operation, you would see then see the child operation.

Let us make a second child operation in order to truly demonstrate the ability of Execution Groups. Create another child operation following the steps above and set its **Parent Operation** to the same as before. Now, run the application again and open the Execution Group.




*Shows all employees in the company.*

## View All Employees



▲ Employee # ▼	▲ First Name ▼	▲ Middle Initial ▼	▲ Last Name ▼	▲ Work Dept. ▼
60700	Rowena	A	Stanley	IT
000001	Erica	J	Piniero	A00
000003	Joe	E	Klocke	D01
000010	Ted	B	Cessna	G22
000011	Paul	M	Thomas	R55
000020	Steve	Q	Beechstreet	H22
000030	John	D	Qu	G22
000050	Paul	M	Tim	C01
000060	Doddy	F	STERN	R44
000070	EVA	L	Jensen	E01

*Search Employees by Department*

Work Dept. =  

*Execution Group 1 displays 2 different operations simultaneously: View All Employees*

The displayed result should be similar to the sample figure above, depending on what your child operations were set to do. In the sample above, the operations used are as of follows:

- Execution Group 1, type: Execution Group
- (Child) View All Employees, type: SQL; `SELECT * FROM PJDATA.EMPLOYEE`
- (Child) Search By Department, type: SQL; `SELECT * FROM PJDATA.EMPLOYEE WHERE WORKDEPT = ?`

## Blob File Upload and Download

[EE] Instead to setting up WOW as a File Server there may be cases where you would actually like to store files inside of the database rather than on the Application Server. WOW can upload any file into a blob field of a database or you can serve documents off of WOW from a database blob field. Some examples include storing/serving contracts, forms, pictures, PDFs and Word documents that are associated with records in the database. Any type of binary file can be served or uploaded into the blob field using WOW. In this example, the `WOWSAMPLES.EMPLOYEEFILES` table will be used.

The table used to store blob entries and other files including attachments must contain at least the required fields marked with a \* below (and have the specified usage ID if needed).

The fields and their possible values are as follows:

- ID – A unique ID assigned to each blob file. \*
- EMPLOYEEENUM – This field is used to match files to the employee table.
- FILENAME – The file name. \* Usage Id: -200
- DESCRIPTION – A description of the file.
- MIME\_TYPE – Type of file specification. \* Usage Id: -190
- FILE\_SIZE – Size of the file being stored in the database. \* Usage Id: -210
- UPLOAD\_TMSP - The timestamp of when the file was uploaded.
- LAST\_DOWNLOAD\_TMSP - The timestamp of when the file was last downloaded.
- FILE\_BLOB – Blob field that stores the file. \*

Any file can be used to store the blob field but it must at least have the required fields and usage IDs set from the files table above.

## Set Up File Upload

[EE] In this example, we will upload some files and associate them with employees from the `pjdata.employee` table. The operation we used, All Employees, selects some basic fields from the employee table and also includes a derived field called `d_upload_files`.



## Web Object Wizard

Development Tools

### First Time User?

Check out this [quick video](#) to get started building a simple application!



### Web Application Creation Steps:

1

[Setup Connection\(s\)](#)

2

[Setup Application\(s\)](#)

3

[Setup Operation\(s\)](#)

Previous

Update and Previous

### Basic

Label\*

All Employees

Title

Operation Type\*

SQL

Description

```
SELECT 'Upload File' as d_upload_files,
FIRSTNAME, LASTNAME, EDLEVEL, IMAGE FROM p...
```

Operation Code

Instructions

Output Connection Alias

Create the derived field descriptor for `d_upload_files`.

## Field Descriptor Manager

Home Shared FDs Usagelds Search By Quick Edit

Log Out

### Basic Settings

Field Name*	D_UPLOAD_FILES	External Name
Required	<input type="checkbox"/>	Required On Search
Default Value	<input type="radio"/> -- None -- <input checked="" type="radio"/> Upload File	Auto Update Value

### Display Settings

Field Set	<input checked="" type="radio"/> <input type="text"/> <input type="radio"/> <input type="text"/>	Display Order
Display Rule*	Always	Display Component*
Help Text	<input type="text"/>	Style Class
Display Width	<input type="text"/>	Display Height

### Possible Value Settings

Possible Values Key	-- None --	Possible Values Operation
Possible Value Class	<input checked="" type="radio"/> -- None -- <input type="radio"/> <input type="text"/>	

Here is the File Upload All Employees operation without associations:

# File Upload and Download Example

Default

## All Employees of PlanetJ



▲ Upload File ▼	▲ EMPNO ▼	▲ First Name ▼	▲ Last Name ▼	▲ EDLEVEL ▼
Upload File	000010	Charles	Turner	12
Upload File	000020	Terry	Sheppard	16
Upload File	000030	John	Qu	18
Upload File	000050	Frank	Tim	25
Upload File	000060	Doddy	STERN	25
Upload File	000070	EVA	Jensen	25
Upload File	000090	EILEEN	PINEIRO	27

After creating the derived field, we need to create FDs for the `employeefiles` table. The `employeefiles` table will hold the files associated with each employee. In the Connections screen, click on the "Edit FD's" link next to the relevant connection and navigate to `wowsamples.employeefiles`. Under table functions, click the "Create FD's" link. Edit the `FILE_NAME`, `MIME_TYPE` and `FILE_SIZE` field descriptors and set their usage IDs.

`FILE_NAME: -200`

`MIME_TYPE: -190`

`FILE_SIZE: -210`

**NOTE:** If going against MySQL, WOW sometimes recognizes BLOB fields as SQL Type `VARBINARY`. For file upload to work correctly, we need to use the `Blob` SQL Type as shown below.

Field Descriptor Type*	Default	Concurrency*
Getter Method		Setter Method
Association Operation	-- None --	Notify Status Change
Remarks		
XML Tag		

### Authorization Settings

Read Authorization Operation	-- None --	Edit Authorization Operation
------------------------------	------------	------------------------------

### Additional Settings

Sortable	<input checked="" type="checkbox"/>	Auto Increment
Read Only	<input type="checkbox"/>	Auto Trim On Read
Currency		Auto Trim On Write
Id*		Usage Id

### Database Settings

System Alias*		Column Size
Library Name*		Scale
Table Name*		Nullable
SQL Type*	BLOB	Key Position
SQL Type Name*	BLOB	

Next, create an associated operation to insert the related files into the database. In this case we have already set the usage ids for FILE\_NAME, MIME\_TYPE and FILE\_SIZE and have set the ID field to auto increment. These fields will be filled in automatically when we try to insert a new file. Now, we need to associate the file with a particular employee, so we will default the employeeenum field to ??EMPNO which fills in each employee's number from the employee table.



Field Descriptor Manager - Powered by PlanetJ's WOW - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

← → ↻ × 🏠

http://localhost:8080/wow64dev/runApp?ACTION\_REQUESTED=FDMGR\_ACTION\_VIEW\_FD&id=%2D9

Getting Started Latest Headlines Mozilla Firefox Start P... Mozilla Firefox Start P...

PLANETJ CORPORATION

Field Descriptor Manager

Usagelds Shared FDs Home Search By Quick Edit Log Out

Update FD

Basic Settings

Field Name\*

EMPLOYEEENUM

External Name

Required

☐

Required On Search

Default Value

☒ -- None --

☒ ??EMPNO

Auto Update Value

Display Settings

Field Set

☒

☐

Display Order

Display Rule\*

Always

Display Component\*

Help Text

Style Class

Display Width

Display Height

Possible Value Settings

Possible Values Key

-- None --

Possible Values Operation

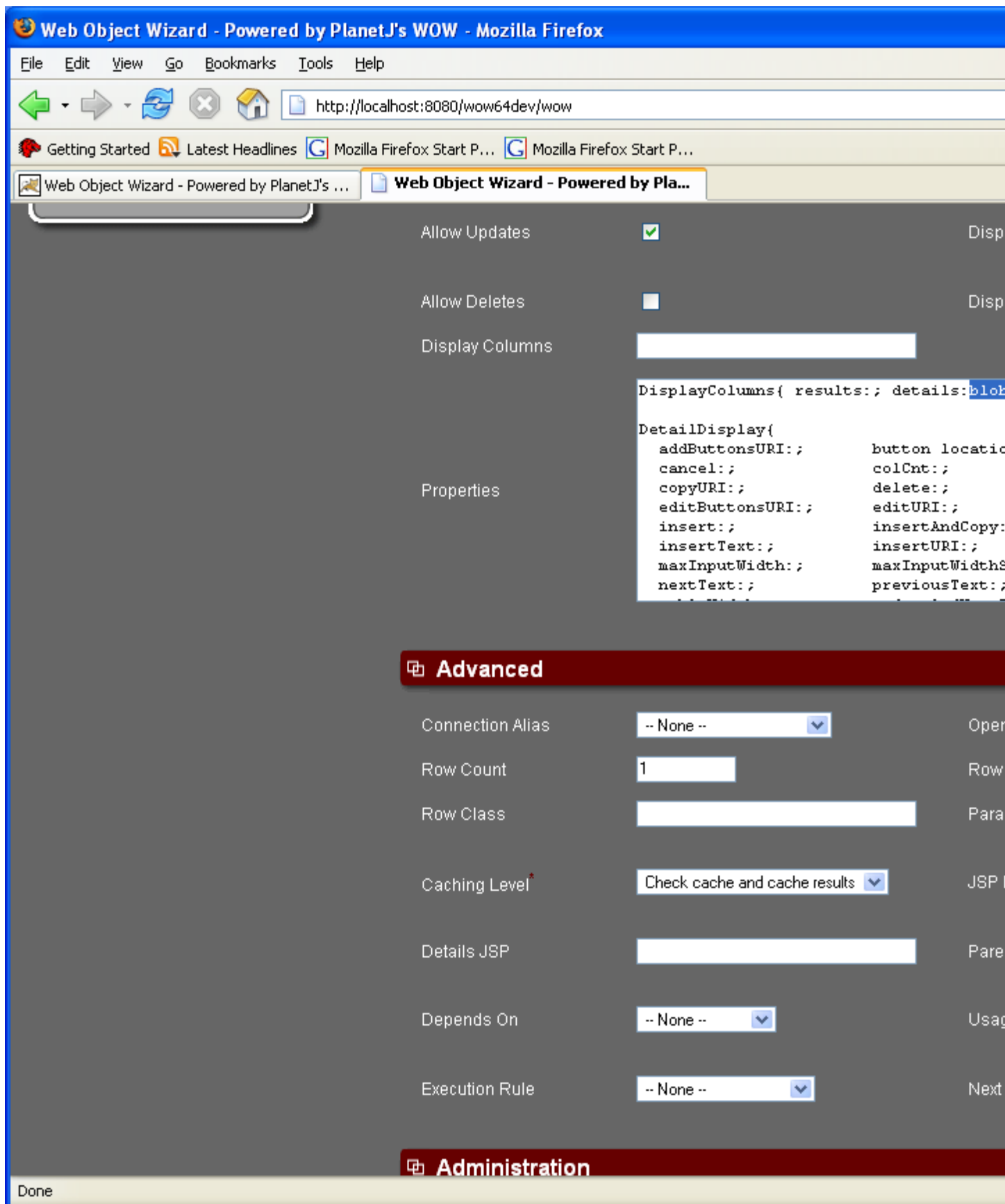
☒ -- None --

Done

The associated insert operation will be a standard insert statement except for a few property changes. First, we only want to insert one file at a time so we need to change the operations row count to 1 instead of 50.



Now that we have set the required usage ids for the other relevant fields, we want to only show the blob\_file field when inserting. To do this we must add the blob\_file to the details property of the DisplayColumns property group: `DisplayColumns{details:blob_file;}`



After we have inserted the operation, we then need to associate with the d\_upload\_files field. Open the d\_upload\_files FD and set the Associated Operation to the upload file association operation that was just created.

Field Descriptor Manager - Powered by PlanetJ's WOW - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

←
→
↻
✕
🏠
http://localhost:8080/wow64dev/runApp?ACTION\_REQUESTED=FDMGR\_ACTION\_VIEW\_FD&id=%2D%

🐉 Getting Started
📰 Latest Headlines
🌐 Mozilla Firefox Start P...
🌐 Mozilla Firefox Start P...

Possible Value Class

☒ -- None --

☐

Advanced Settings

Field Class

☒ -- None --

☐

Field Descriptor Type\*

Derived

Getter Method

Association Operation

File Upload

Remarks

XML Tag

Formatter Class

Concurrency\*

Setter Method

Notify Status Change

Authorization Settings

Read Authorization Operation

-- None --

Edit Authorization Operation

Additional Settings

Sortable

☒

Read Only

☐

Currency

☐

Id\*

122405

Auto Increment

Auto Trim On Read

Auto Trim On Write

Usage Id

Done

Run the Employee operation and click on the "Upload File" link.



File Upload and Download Example - Powered by PlanetJ's WOW - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://localhost:8080/wow64dev/runApp

Getting Started Latest Headlines Mozilla Firefox Start P... Mozilla Firefox Start P...

# File Upload and Download Example

Default

## All Employees of PlanetJ

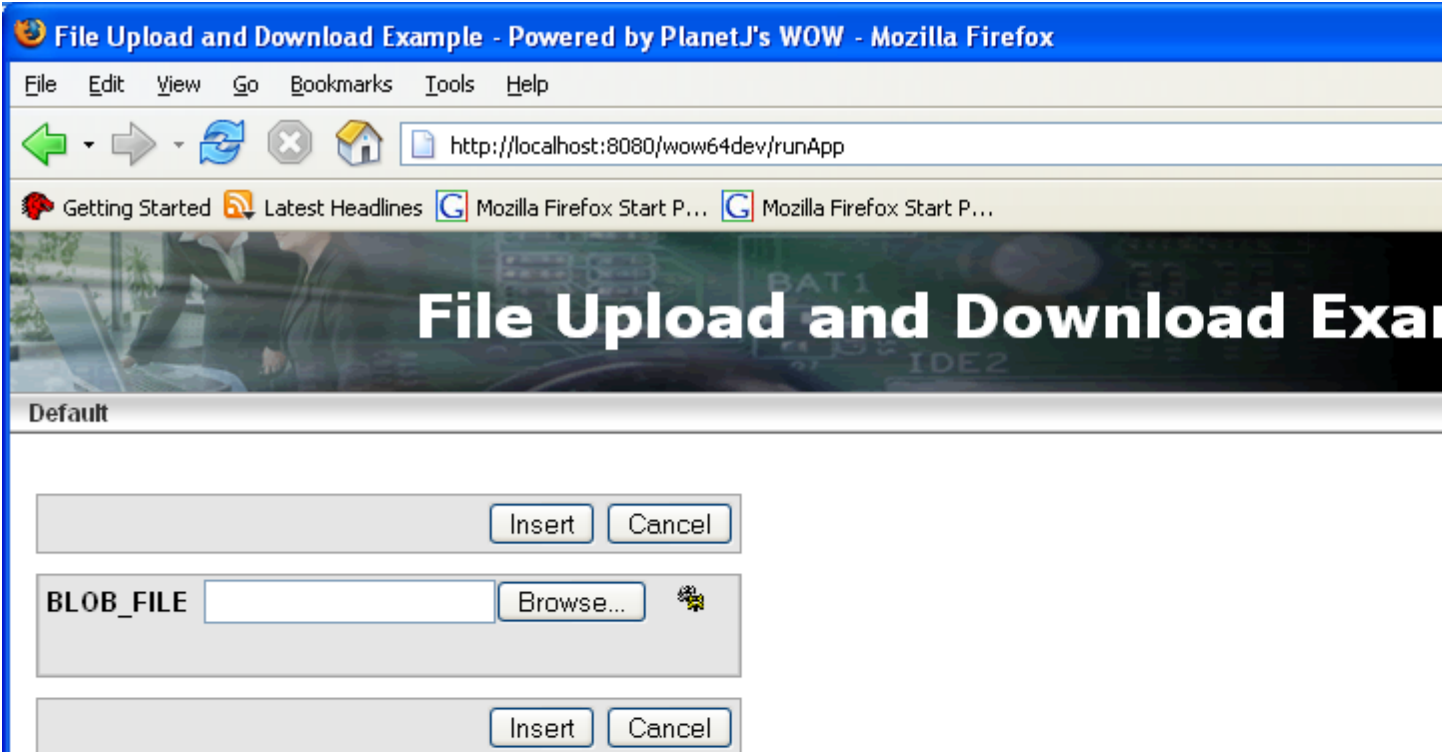
▲ Upload File ▼	▲ Employee # ▼	▲ First Name ▼	▲ Last Name ▼	▲ EDLEVEL ▼
Upload File	000001	Erica	Piniero	18
Upload File	000003	Laura	Klocke	12
Upload File	000010	Ted	Cessna	16
Upload File	000011	Paul	Thomas	16
Upload File	000020	Steve	Beechstreet	16
Upload File	000030	John	Qu	18
Upload File	000050	Paul	Tim	25
Upload File	000060	Doddy	STERN	25
Upload File	000070	EVA	Jensen	25
Upload File	000090	Eileen	Piniero	25
Upload File	000100	Paul	SPENSER	14
Upload File	000110	Al	Gregga	10
Upload File	000120	SEAN	O'CONNELL	14
Upload File	000130	DELORES	QUINTANA	30
Upload File	000140	HEATHER	NICHOLLS	26
Upload File	000150	Don	Jesik	25
Upload File	000152	Gerald	BOB	25
Upload File	000160	ELIZABETH	PIANKA	26
Upload File	000190	Charlena	WALKER	25
Upload File	000200	DAVID	BROWN	25
Upload File	000210	WILLIAM	JONES	26
Upload File	000220	JENNIFER	LUTZ	16
Upload File	000230	JAMES	JEFFERSON	14
Upload File	000240	SALVATORE	MARINO	26
Upload File	000250	DANIEL	SMITH	24
Upload File	000260	SYBIL	JOHNSON	25
Upload File	000270	MARIA	PEREZ	24
Upload File	000290	JOHN	PARKER	12
Upload File	000300	PHILIP	SMITH	14
Upload File	000320	RAMLAL	MEHTA	25
Upload File	000322	Jon	CNote	78
Upload File	000330	John	LEE	14
Upload File	000340	JASON	GOUNOT	25

Done

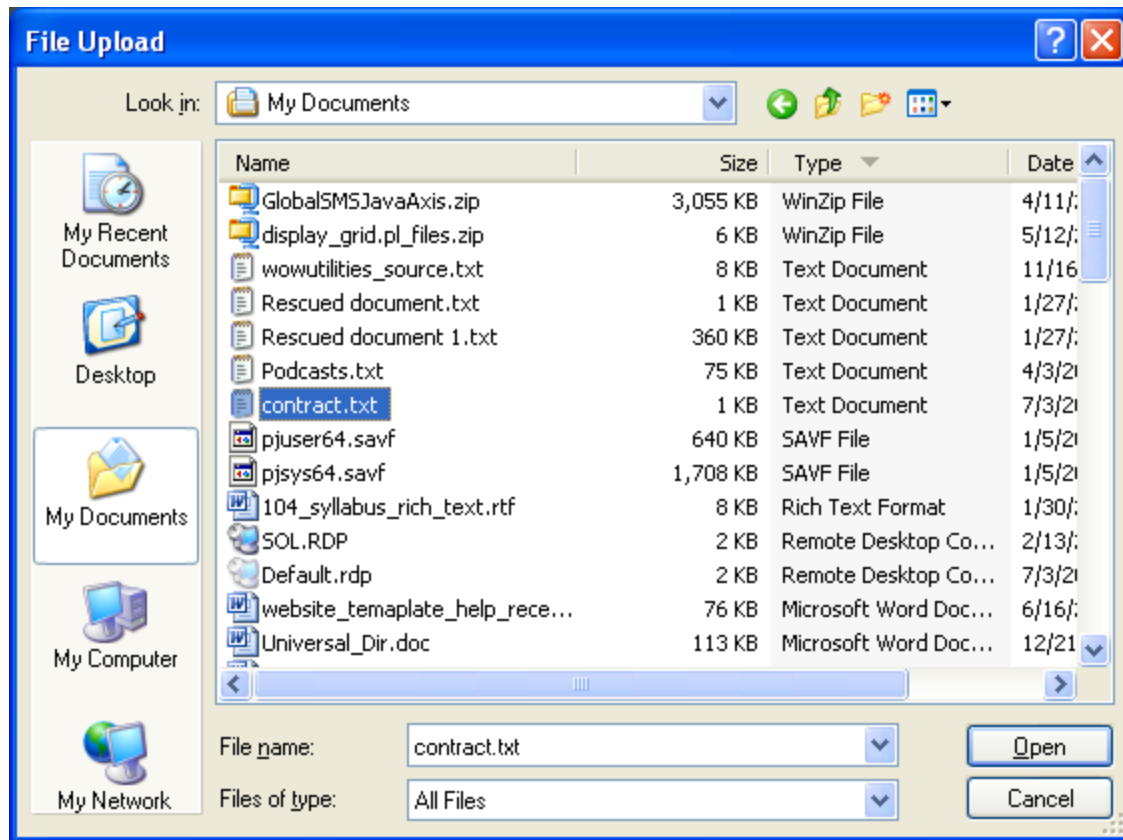
Use the browse button to find the file, image, or document that you would like to upload to the blob field and then click the insert button. If your BLOB\_FILE field does not have the [Browse...] button next to it, check the FD and ensure that the Display Component is set to "File Upload" as in the image below:



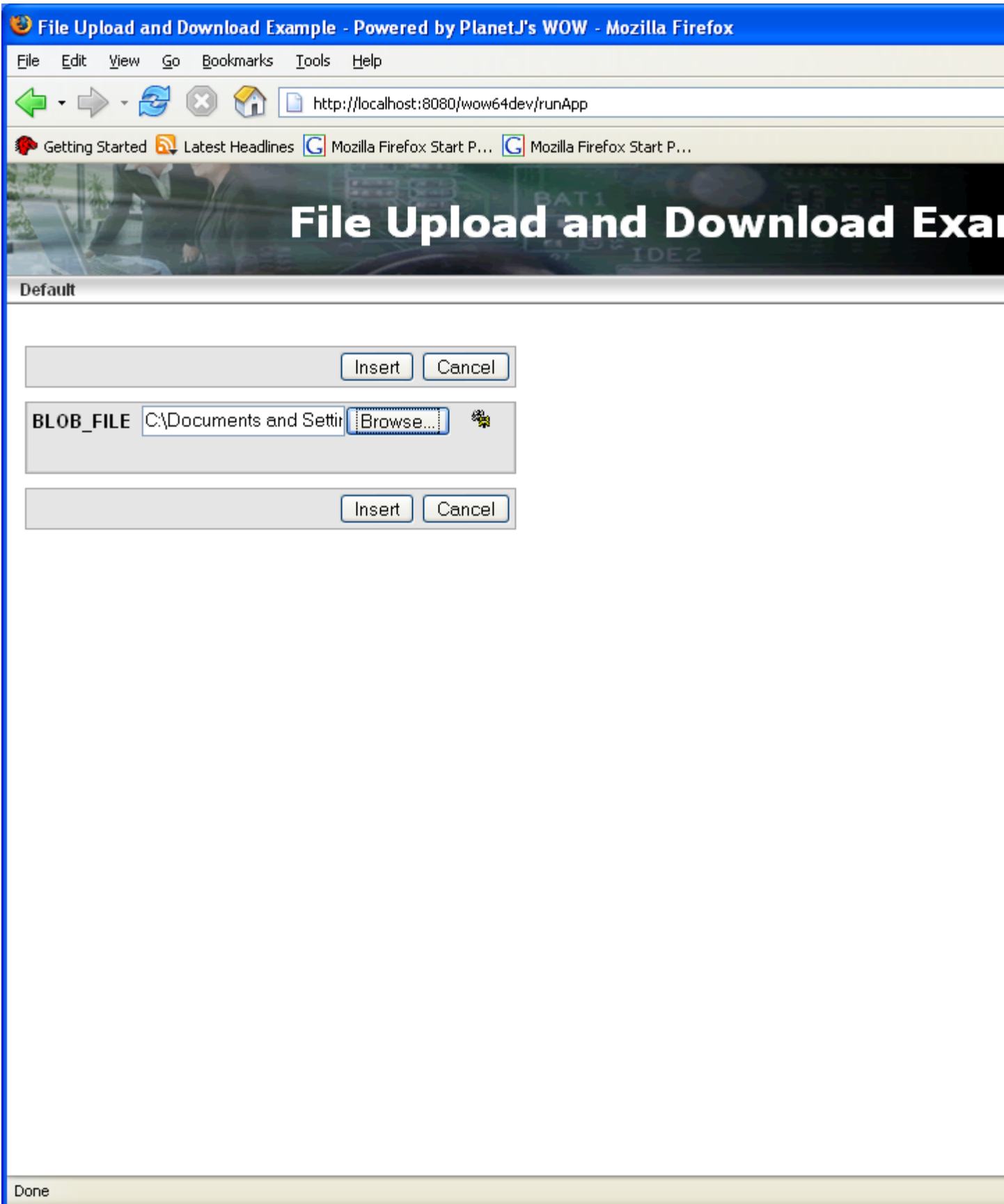
Now, you will have a file that is associated with the employee and stored in a blob field.



Here is the blob file upload file explorer screen.



Here is the filled blob file upload field.



## Set Up File Download

Now that we have setup file upload, we also need to have the ability to download or open those files from the database. In this example, we will edit the All Employees operation and add another derived FD called d\_view\_files.

```
SELECT 'Upload File' AS d_upload_files, 'View Files' AS d_view_files, empno,  
firstnme, lastname, edlevel, image FROM pjdata.employee
```



Now, we need an associated operation which will show all the files associated with a particular employee. Create an Association 1-Many operation to show all files associated with the selected employee.

```
SELECT * FROM wowsamples.employeefiles WHERE employeenum = ??empno
```





Associate this operation with d\_view\_files FD. Then, run the All Employees operation. Now there is an "All Employee Files" link which can be clicked on to see all of the employees' files.

File Upload and Download Example - Powered by PlanetJ's WOW - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://www.planetjavainc.com/wow400/runApp

Getting Started Latest Headlines Mozilla Firefox Start P... Mozilla Firefox Start P...

# File Upload and Download Example

Default

## All Employees of PlanetJ

▲ Upload File ▼	▲ All Files ▼	▲ EMPNO ▼	▲ First Name ▼	▲ MIDINIT ▼	▲ Last Name ▼
<a href="#">Upload File</a>	<a href="#">All Employee Files</a>	000010	john	B	doe
<a href="#">Upload File</a>	<a href="#">All Employee Files</a>	000020	Terry	Q	She
<a href="#">Upload File</a>	<a href="#">All Employee Files</a>	000030	John	C	Qu
<a href="#">Upload File</a>	<a href="#">All Employee Files</a>	000050	Frank	M	Tim
<a href="#">Upload File</a>	<a href="#">All Employee Files</a>	000060	Doddy	F	STE
<a href="#">Upload File</a>	<a href="#">All Employee Files</a>	000070	EVA	L	Jens
<a href="#">Upload File</a>	<a href="#">All Employee Files</a>	000090	EILEEN	W	PINE

After viewing all files, we need to create a file download operation to actually download the

file to the local computer from the blob field. Create a new operation of type File Download and select the relevant record from the employeefiles table.

```
SELECT * FROM wowsamples.employeefiles WHERE id = ??id
```

## Web Object Wizard

Development Tools

### First Time User?

Check out this [quick video](#) to get started building a simple application!



### Web Application Creation Steps:

1

[Setup Connection\(s\)](#)

2

[Setup Application\(s\)](#)

3

[Setup Operation\(s\)](#)

4

[Run!](#)

### Basic

Label

Download File Assoc.

Title

Operation Type

Blob Download



Description

```
SELECT * FROM wowsamples.employeefiles
```

Operation Code

Instructions

Output Connection Alias

### Display

Allow Details



Display

Now that the download file association is created, we can associate it with the "View Employees Files" operation, specifically, the file\_name FD.

Display Width

Display Height

## Possible Value Settings

Possible Values Key

Possible Values Operation

Possible Value Class

☒ -- None --

☐

## Advanced Settings

Field Class

☒ -- None --

☐

Formatter Class

Field Descriptor Type\*

Concurrency\*

Getter Method

Setter Method

Association Operation

Notify Status Change

Remarks


XML Tag

## Authorization Settings

Read Authorization Operation

Edit Authorization Operation

## Additional Settings

Sortable

☒

Auto Increment

Read Only

☐

Auto Trim On Read

Currency

☐

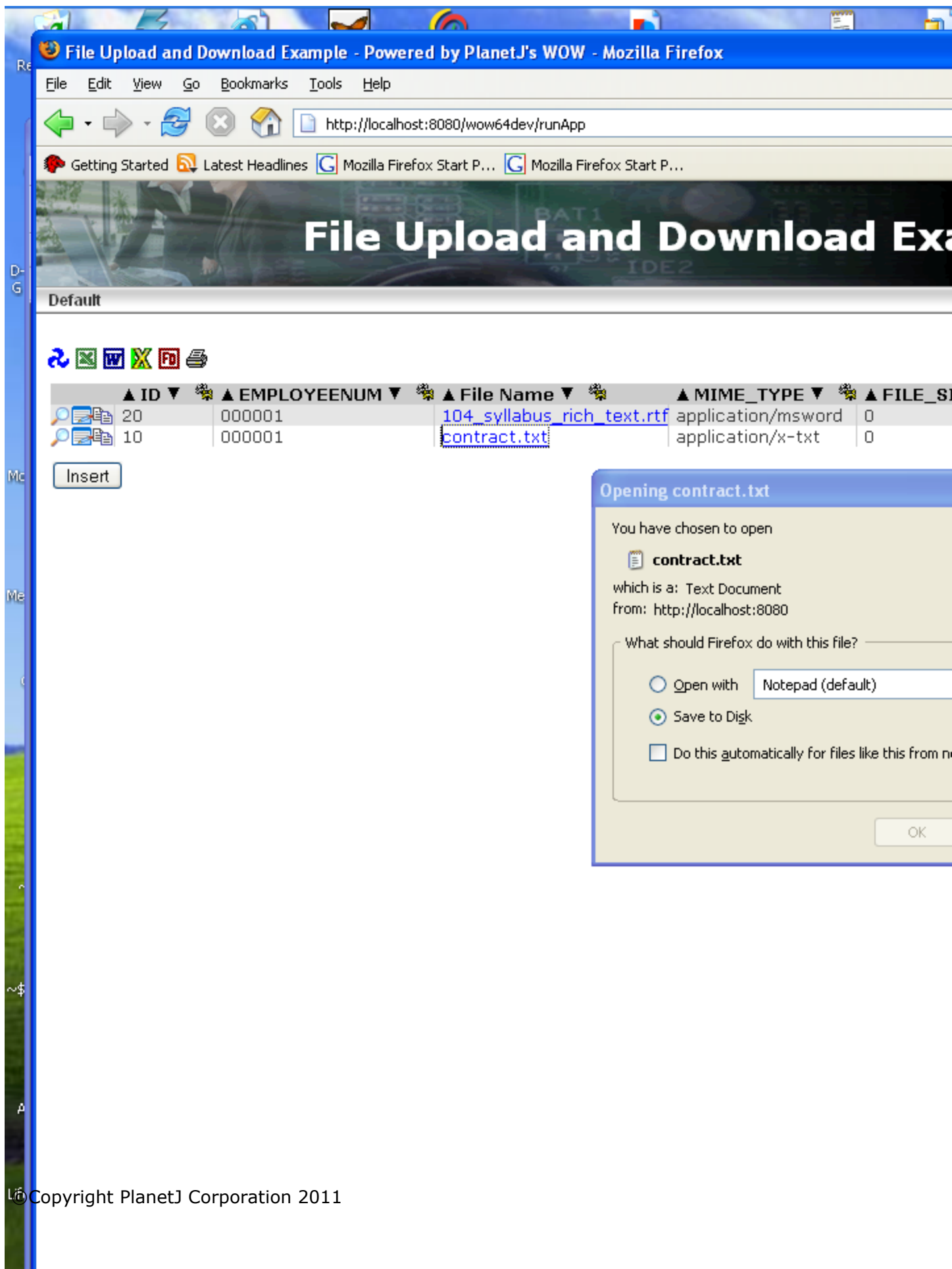
Auto Trim On Write

Id\*

Usage Id

Now, when a user clicks on the field name they will be prompted to download the file to their local file system.



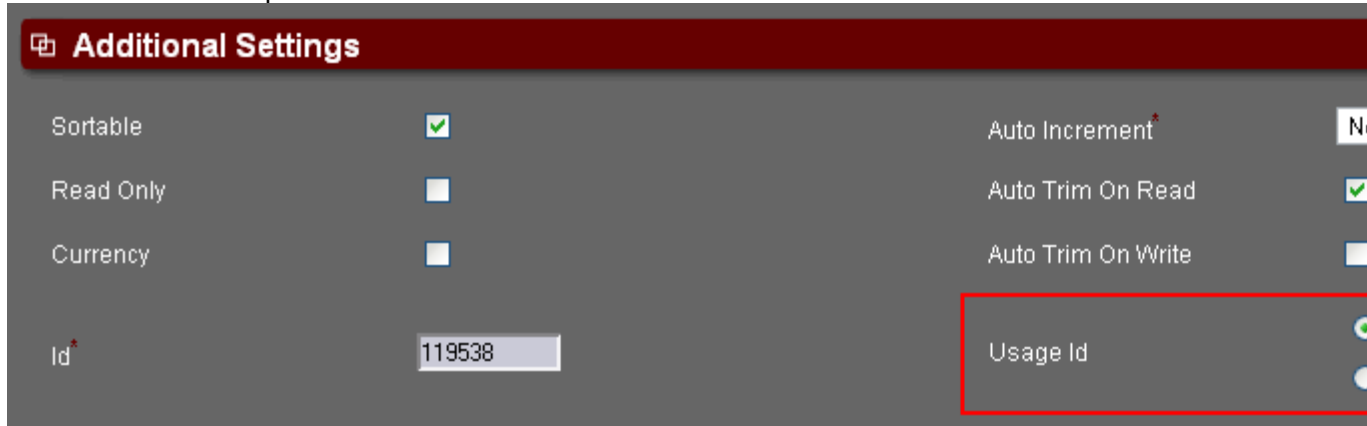


## Work Flow

Traditional WOW development easily handles operations such as data lookups, edits, deletes, and provides some "work flow" via associated operations. More complex applications require real "work flow". Work flow is the logical transition through a number of screens. The sequence of screens could lead to some composite transaction at the end of the process. An example would be an application that enables making an airline reservation. To enable this capability, two new features have been created:

- **Global Variables**

Variables that can be created and shared throughout a user's session. Global variables are established and created by setting the field descriptor's UsageId to -3. When WOW processes an associated selection, insert, update, or delete of a row, any fields present that are marked as Global are set in the user's session and available to other operations. Other operations may reference those global variables using "??!" followed by the field name (e.g. ??!orderNum). Global variables remain in effect until a row containing that variable is selected, updated, inserted, or deleted and the new row field values override the previous values. Global values can also be references in another field descriptor's default value.



Additional Settings	
Sortable	<input checked="" type="checkbox"/>
Read Only	<input type="checkbox"/>
Currency	<input type="checkbox"/>
Id*	119538
Auto Increment	<input type="checkbox"/>
Auto Trim On Read	<input checked="" type="checkbox"/>
Auto Trim On Write	<input type="checkbox"/>
Usage Id	<input checked="" type="checkbox"/>

- **Next Operation**

WOW operations now have a field called "Next Operation" which allows the selection of another operation to execute when the current operation completes. The current operation completes when a row is inserted, updated, or deleted. At that time, the next operation is executed. The next operation may, in turn, have a next operation specified, thus enabling a complete flow of an application without the need to manually program.

Advanced

Connection Alias	-- None --	Operation Class	
Row Count	50	Row Coll. Class	
Row Class		Parameters JSP	
Caching Level	Check cache and cache results	JSP File	<input checked="" type="radio"/> -- None -- <input type="radio"/>
Details JSP		Parent Operation	-- None --
Depends On	-- None --	Usage Id	<input checked="" type="radio"/> -- None -- <input type="radio"/>
Execution Rule	-- None --	Next Operation	Confirmation Page

### Example 1

A common business scenario involves creating orders by first inserting an order header such as customer name, order number, address, date, etc. Upon completion of the order header record, the user is then allowed to enter order detail records. In this example, the WOW developer would specify the order number (in the order header) as a global variable by setting its field descriptor to -3. Assuming both order header and order detail files have an order number field, the order number field (in the order detail file) would have its field descriptor's default value set to "?!OrderNumber" where OrderNumber is the field in the order header file. When each order detail record is created, its order number is automatically set to the value in the order header (which is the global variable set in the current session). This allows the complete ordering process to be carried out cohesively. Each new order would get a new order number and each order detail would be tied to its order header.

### Example 2

Another common scenario for the work flow feature is a confirmation page. For instance, if an end user inserts a bug report record, it may be desired to show a confirmation page assuring the user that his entry has been accepted and will be acted upon. In this case, mark any fields in the bug report's field descriptors as global by setting the UsageId to -3. Create a HTML code operation that includes confirmation text as well as the desired global variables. For example:

"Your bug report has been logged and the ID is: ?!problemNum"

## Advanced Work Flow

Sometimes, more complicated work flow scenarios may require programmatic control in terms of what the next completed operation should be. For example, an international flight reservation may require a different sequence of steps (operations) then a domestic

flight. Or, if an insurance quote is inserted and it is more than 10,000,000 then it requires additional info, etc. To accommodate these advanced work flow needs, the WOW Java based framework can be overridden allowing you to control the exact flow. For more information, see the section on work flow in the WOW Programmer's Guide.

# Context Menu

## Controlling Actions in the Context Menu

There are many ways that the context menu can be modified. This will be described in steps below.

### Disabling the Context Menu

If the *contextMenu* property is set to false in the TableDisplay then the entire context menu will never be displayed for any rows in that table.

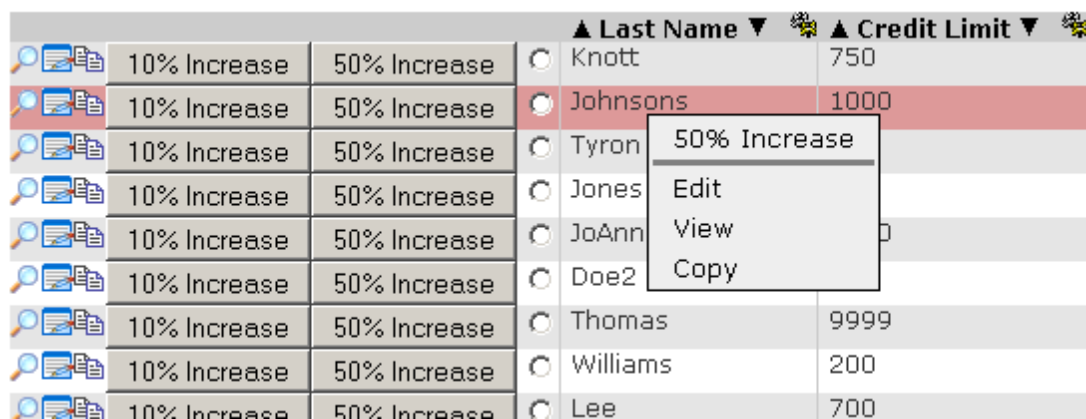
```
TableDisplay{  
    contextMenu: false;  
}
```

### Removing Actions from the Context Menu

By default the context menu will contain all of the rows' actions. If you do not wish an action to show in the context menu, set the *contextMenu* property to false in that action's ActionDescriptor property group. For example, an operation with the following property groups:

```
ActionDescriptor{  
    name: CDT10;  
    actType: row;  
    contextMenu: false;  
    label: 10% Increase;  
    dspOrder: 10;  
}  
ActionDescriptor{  
    name: CDT50;  
    actType: row;  
    label: 50% Increase;  
    dspOrder: 20;  
}
```

would have a context menu like this:



The screenshot shows a table with columns for actions and data. The actions are '10% Increase' and '50% Increase'. The data columns are 'Last Name' and 'Credit Limit'. A context menu is open over the '50% Increase' action for the row 'Tyron', showing options: '50% Increase', 'Edit', 'View', and 'Copy'. The '10% Increase' action is not visible in the context menu.

			▲ Last Name ▼	▲ Credit Limit ▼
10% Increase	50% Increase	<input type="radio"/>	Knott	750
10% Increase	50% Increase	<input type="radio"/>	Johnsons	1000
10% Increase	50% Increase	<input type="radio"/>	Tyron	
10% Increase	50% Increase	<input type="radio"/>	Jones	
10% Increase	50% Increase	<input type="radio"/>	JoAnn	
10% Increase	50% Increase	<input type="radio"/>	Doe2	
10% Increase	50% Increase	<input type="radio"/>	Thomas	9999
10% Increase	50% Increase	<input type="radio"/>	Williams	200
10% Increase	50% Increase	<input type="radio"/>	Lee	700

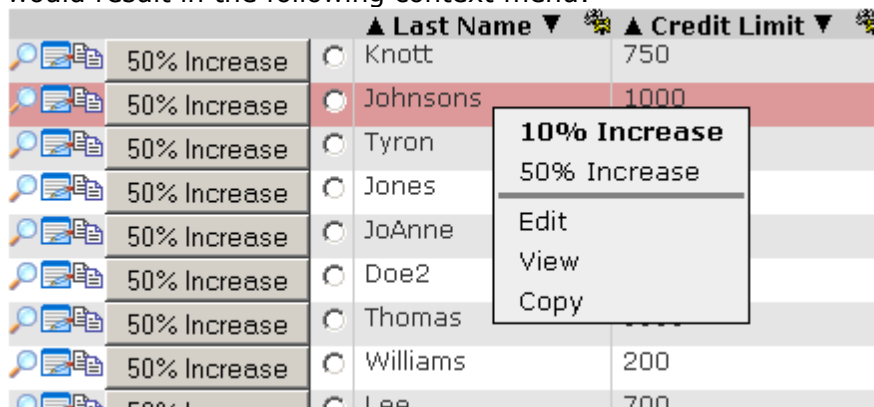
Notice the 10% Increase action is not included in the context menu.

## Showing Actions Only in the Context Menu

You can use the *loc* property of the ActionDescriptor property group to show actions only in the context menu. If the *loc* property is not present or is left blank then WOW will show the action both inline in the row as well as in the context menu. If the location "context menu" is the only location specifically listed in the *loc* property, then WOW will show the action in the context menu, and will not show the action in any other location. So an operation with these property groups:

```
ActionDescriptor{
  name: CDT10;actType: row;
  loc: context menu;
  label: 10% Increase;
  dspOrder: 10;
}
ActionDescriptor{
  name: CDT50;
  actType: row;
  label: 50% Increase;
  dspOrder: 20;
}
```

would result in the following context menu:



Notice that the 10% Increase button does not appear in the row actions, only on the context menu.

## Suppressing Built-in Actions

Normally WOW will include applicable built-in actions in the context menu such as View, Edit, and Delete. These actions will only show if the row supports that action. (For example the context menu will not contain a Delete action if the row cannot be deleted.) However you can also suppress these built-in actions using the ActionContextMenuDescriptor property group. The ActionContextMenuDescriptor property group is a property group which applies only to actions in the context menu. All properties in the ActionDescriptor property group are also present in the ActionContextMenuDescriptor property group, except for the *loc* property (since the context menu is the only location to which that property group applies). Setting the *dspTyp* property to "none" in the ActionContextMenuDescriptor property group will prevent the built-in action from being displayed in the context menu.

In order to hide the "Edit" in context menu:

		▲ Last Name ▼	▲ Credit Limit ▼
	50% Increase	<input type="radio"/> Knott	750
	50% Increase	<input checked="" type="radio"/> Johnsons	1000
	50% Increase	<input type="radio"/> Tyron	
	50% Increase	<input type="radio"/> Jones	
	50% Increase	<input type="radio"/> JoAnne	
	50% Increase	<input type="radio"/> Doe2	
	50% Increase	<input type="radio"/> Thomas	
	50% Increase	<input type="radio"/> Williams	200
	50% Increase	<input type="radio"/> Lee	700

**10% Increase**  
50% Increase  


---

Edit  
View  
Copy

the following property group could be put into the operation properties:

```

    ActionContextMenuDescriptor{
    name: edit;
    actType: row;
    dspType: none;
    }

```

Now the row's context menu will not contain the Edit action:

		▲ Last Name ▼	▲ Credit Limit ▼
	10% Increase	<input type="radio"/> Knott	750
	10% Increase	<input checked="" type="radio"/> Johnsons	1000
	10% Increase	<input type="radio"/> Tyron	
	10% Increase	<input type="radio"/> Jones	
	10% Increase	<input type="radio"/> JoAnne	
	10% Increase	<input type="radio"/> Doe2	
	10% Increase	<input type="radio"/> Thomas	9999
	10% Increase	<input type="radio"/> Williams	200
	10% Increase	<input type="radio"/> Lee	700

**10% Increase**  
50% Increase  


---

View  
Copy

## Controlling the Context Menu Appearance

There are many situations in which you may want to control the appearance of the context menu. We will go through some of these scenarios in depth here in this chapter.

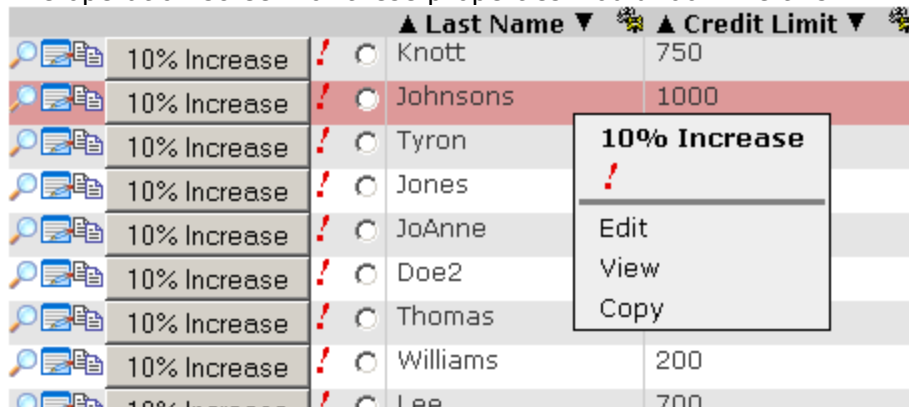
### Using Different Action Descriptors

In some cases you may have a single action which should be displayed in a certain way in the context menu, and displayed a different way inline in the row.

Consider these property groups:

```
ActionDescriptor{
  name: CDT10;
  actType: row;
  label: 10% Increase;
  dspOrder: 10;
}
ActionDescriptor{
  name: WARN;
  actType: row;
  dspType: link;
  imgsrc: /dataengine/images/epoint.gif;
  dspOrder: 20;
}
```

The operation screen for these properties would look like this:



The screenshot shows a table with columns for 'Last Name' and 'Credit Limit'. Each row has a '10% Increase' link in the first column and a radio button in the second column. A context menu is open over the '10% Increase' link in the row for 'Tyron', showing the text '10% Increase' with a red exclamation mark icon, and three options: 'Edit', 'View', and 'Copy'.

	▲ Last Name ▼	▲ Credit Limit ▼
10% Increase	<input type="radio"/> Knott	750
10% Increase	<input type="radio"/> Johnsons	1000
10% Increase	<input type="radio"/> Tyron	
10% Increase	<input type="radio"/> Jones	
10% Increase	<input type="radio"/> JoAnne	
10% Increase	<input type="radio"/> Doe2	
10% Increase	<input type="radio"/> Thomas	
10% Increase	<input type="radio"/> Williams	200
10% Increase	<input type="radio"/> Lee	700

The same gif is used to display the WARN action in the row and in the context menu, and the same text is used to display the "10% Increase" action in both places. Using the ActionContextMenuDescriptor property group you can set display properties which apply only to the context menu. When a ActionContextMenuDescriptor is present for an action, then the ActionDescriptor property group is not used to render that action in the context menu. With these property groups:

```
ActionDescriptor{
  name: CDT10;
  actType: row;
  label: +10%;
  dspOrder: 10;
}
ActionDescriptor{
  name: WARN;
  actType: row;
  dspType: link;
```












```

imgsrc: /dataengine/images/epoint.gif;
dspOrder: 20;
}
ActionContextMenuDescriptor{
name: CDT10;
actType: row;
dspType: text;
label: 10% Increase;
}
ActionContextMenuDescriptor{
name: WARN;
actType: row;
dspType: text;
label: Send credit warning;
}
}

```

The screen now looks like this:

			▲ Last Name ▼	▲ Credit Limit ▼
	+10%	!	Knott	750
	+10%	!	Johnsons	1000
	+10%	!	Tyron	
	+10%	!	Jones	
	+10%	!	JoAnne	
	+10%	!	Doe2	
	+10%	!	Thomas	
	+10%	!	Williams	200
	+10%	!	Lee	700

**10% Increase**

Send credit warning

---

Edit

View

Copy

## CSS Properties

The following CSS Properties are used to control the appearance of the context menu:

- **actionmenu** - Applied to both TABLE and DIV elements which comprise the context menu. (The TABLE is contained in the DIV)
- **actMI** - Applied to each TR in the context menu. There is one TR for each action.
- **actDefMI** - Applied to the TR in the context menu which represents the action which is the default action for the row.
- **actMI-hlight** - Used for the highlighted TR element, which represents the action within the context menu over which the mouse is hovering.
- **actSep** - Used for the TR element which acts as a separator between different groups of actions.
- **hidden** - Applied to the TR element for actions which are in the context menu but cannot be run.

To change the look and feel of the context menu you can change/replace the definition of these CSS styles.

## Action Groups

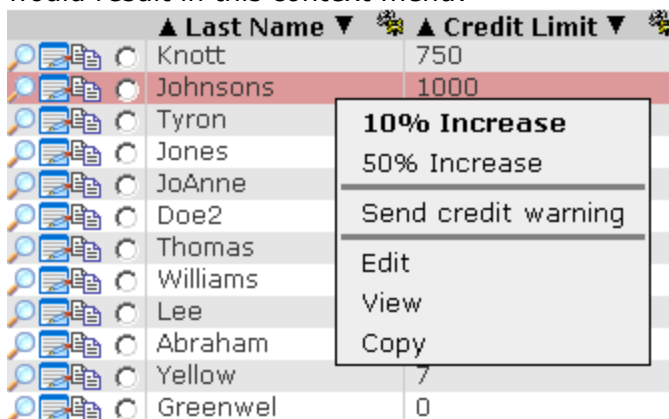
The *group* property in the ActionDescriptor and ActionContextMenuDescriptor property groups can be used to assign multiple actions to a single group. The context menu will show a separator between different groups of actions. These property groups:

```

ActionDescriptor {
name: CDT10;
actTyp: row;
label: 10% Increase;
loc: context menu;
group: credit limit;
dspOrder: 10;
}
ActionDescriptor {
name: CDT50;
actTyp: row;
label: 50% Increase;
loc: context menu;
group: credit limit;
dspOrder: 20;
}
ActionDescriptor {
name: WARN;
actTyp: row;
loc: context menu;
label: Send credit warning;
dspOrder: 30;
}
}

```

would result in this context menu:















Notice the actions divided into 3 different groups, the "credit limit" group, the send credit warning action and the common row actions (Edit, View, Copy).

## Different Actions for Different Rows

If you are writing custom code then you can have different actions appear in the context menu depending on which row the user clicks on.











The actions which may appear in the context menu are determined by the first row of the results. An action may not appear in the context menu unless it is an action for that first row. However, if an action is not applicable for a particular row then it is not shown in the context menu for that row. The `Row.isActionApplicable(String, ExecutingContext)` method is used to determine whether or not an action is applicable to a particular row. If this method is overridden to return false for certain rows, then for those rows only the action will not be displayed in the context menu.

All actions are applicable for this row:

	▲ Last Name ▼	▲ Credit Limit ▼
	Knott	750
	Johnsons	1000
	Tyron	
	Jones	
	JoAnne	
	Doe2	
	Thomas	
	Williams	
	Lee	
	Abraham	
	Yellow	7
	Greenwel	0

- 10% Increase
- 50% Increase
- Send credit warning
- Edit
- View
- Copy

Some actions are not applicable, and therefore are not displayed, for this row:

	▲ Last Name ▼	▲ Credit Limit ▼
	Knott	750
	Johnsons	1000
	Tyron	5000
	Jones	
	JoAnne	
	Doe2	
	Thomas	
	Williams	
	Lee	700
	Abraham	9999

- Send credit warning
- Edit
- View
- Copy

Notice that the second customer does not have the Credit Limit Increase options in the context menu.

# Auto Complete

Fields in WOW can be configured to be "auto complete" fields. When a user begins typing information into an auto complete field, WOW will show a drop down containing the possible values which could complete the user's typing, and highlight the first possible completion. For example, a user may be searching on a state value as shown below:

State =

When the user types an 'M', all the states beginning with the letter 'M' are shown. The first match (Maine) is highlighted.

State =

- Maine
- Maine
- Maryland
- Massachusetts
- Michigan
- Minnesota
- Mississippi
- Missouri
- Montana

The user continues to type, entering an 'i' after the 'M'. Only states beginning with 'Mi' are shown

State =

- Michigan
- Michigan
- Minnesota
- Mississippi
- Missouri

Next, the user types an 'n'. The only matching state (Minnesota) is shown in the drop down and has been auto completed in the search field.

State =

- Minnesota
- Minnesota

At any time during the typing, the user could also use the arrow keys or mouse to select a different value from the drop down to populate the search field with. Using an auto complete field can be useful if there are too many possible values to display in a normal possible values drop down, or if you want to show the user the possible values for the field while still allowing the user to enter a new value which is not among the existing possible values.

## Configuring Auto Complete Fields

In this section, we will configure an existing search field as an auto complete field. We will start out with a simple SQL query which asks the user to enter in a state to search on. Here is the SQL operation:

Basic			
Label	<input type="text" value="County Search"/>	Title	<input type="text" value="Counties"/>
Operation Type *	<input type="text" value="SQL"/>	Description	<input type="text" value="Search for"/>
<pre>SELECT * FROM jetemp.COUNTY WHERE CNTSTATE = ?</pre>			

The CNTSTATE column in the database contains state abbreviations (two characters). So in order to find any matches, the user will have to enter in a matching state abbreviation.

This is what the search screen looks like before auto complete is configured:

State =

To use the auto complete feature, we must first create a possible values operation for the auto complete field. This operation is responsible for taking what the user has entered in the search field and returning the matching possible values. The returned possible values can contain both an internal value (in the first column) and a display value (in the second column). For example, when the user is searching for a state, the internal value may be "CA" and the display value would be "California". Here is what our example possible values operation looks like:

Basic			
Label	<input type="text" value="AC State PV"/>	Title	
Operation Type *	<input type="text" value="Possible Values"/>	Description	
<pre>SELECT STCODE, STNAME FROM JETEMP.STATES WHERE  STNAME like ??CNTSTATE or cast(??1 as CHAR(10)) IS NULL  order by STNAME</pre>			

The possible values operation will search a file containing both the state abbreviations and the full state names, based on a partial state name entered by the user. This partial state name will be pulled from the CNTSTATE field, which is the field that will be displayed on the screen by the main search query. All possible values operations for auto complete fields should use a LIKE comparison in the SQL, since the goal is to find all matches which begin with a value entered by the user.

Our possible values operation returns the state abbreviation in the first column, since that is the internal value required by our primary search. The second column contains the display value (the full state name). The internal value will never be displayed on the screen - the display value is shown in both the drop down and the search field. When data is sent to the database however, WOW will always use the internal value and not the display value.

The next step is to configure the field descriptor for the auto complete field, which is the CNTSTATE field in our example. The display component should be set to Auto Complete, and we must also select the possible values operation we created. In this example we also need to adjust the field size from 2 to 15. The field size was set to 2 when WOW created the field descriptor, since the column holding the state abbreviations in the database can only hold 2 chars. However we want to allow the field to contain the full state name and not just the abbreviation, so we have to increase the field size. This only affects the field within WOW - the database table can still only hold 2 characters.

FIELD NAME : CNTSTATE

Required: ☒ Required On Search: ☐

Default Value:

Auto Update Value:

---

**Display Settings**

Field Set: ☐   
☐

Display Rule:

Help Text:

Display Width:

Display Order:

Display Component:

Style Class:

Display Height:

---

**Possible Value Settings**

Possible Values Key:

Possible Values Operation:

Possible Value Class: ☐ Enter Class Name:   
☐ Select Existing Class:

---

**Database Settings**

Library Name:

Table Name:

System Alias:

SQL Type:

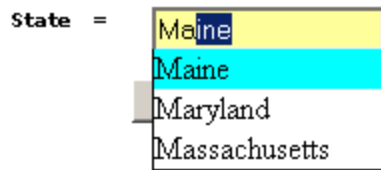
Column Size:

Scale:

Nullable:

Key Position:

After making the above changes to the field descriptor, our auto complete field is now ready to go. We can run the search, and as we type values into the search field WOW will run the possible values operation to retrieve the possible values matching what has been entered in the field and display them in the drop down.



## Auto Complete Properties [PRO]

Once you have an auto complete field configured, you can further customize it using an AutoComplete property group. WOW will look for AutoComplete properties first in the possible values operation for the auto complete field, then in the main query operation, and finally in the application. (AutoComplete properties can be specified in any of these locations, but properties in the possible value operation will take precedence over those in main operation, and properties in the application are overridden by properties from either of the other locations.)

The properties available in the AutoComplete property group are listed below. In general, you can use an AutoComplete field without setting any of these properties – you only need to specify them if you want to change the default setting.

- **cache timeout** – The number of seconds items are retained in the special auto complete cache. This cache is intended to reduce the number of times WOW needs to query the database when the user is typing multiple characters at once. (For example, if we have already searched for states beginning with 'M' and a half second later we are now searching for states beginning with 'Mi', it will be quicker to search among the previously retrieved states than to run another database query.) The default is 15 seconds. (Setting this property to -1 will turn off the special auto complete cache.)
- **case sensitive** – Whether or not the auto complete search is case sensitive. The default is false. Case sensitivity in auto complete searches also depends on the database connection settings for LIKE comparisons. If the connection setting does not match the auto complete setting, then auto complete searches will return inconsistent results.
- **count** – The maximum number of values to display in the drop down. This defaults to the Row Count of the possible values operation. Using a higher Row Count can be beneficial since it allows WOW to retain more values in its cache, which can improve performance. So in some cases you may want WOW to read and cache more possible values than you want to display on the screen, which is why you would adjust the count property.
- **css** – This is the CSS class which is used to display the drop down. The default value is "pjAutoComplete".
- **focus** – Whether or not the drop down can appear when the field gains focus, or if the user has to actually type in order for the drop down to appear. The default is false (meaning the user has to type) unless the min chars property is set to 0, in which case the focus property defaults to true. If the focus property is explicitly set, the min chars setting does not matter.
- **item css** – The CSS class used to display individual items in the drop down list. (The items are rendered as HTML DIV elements). The default value is "pjACItem".
- **highlight css** – The CSS class used to display the highlighted item in the drop down list. (The items are rendered as HTML DIV elements). The default value is "pjACItemH".
- **min chars** – The minimum number of characters which must be present in the search field before the drop down is displayed. The default is 1. This property is related to the focus property.
- **strict** – This is a comma separated list of modes where WOW will enforce strict possible value behavior on the auto complete field. When strict behavior is being enforced, WOW will display an error message if the user enters a value which is not among the possible values. The possible modes for this property are copy, edit, insert, and search. You can also use the special NONE value. If this property is not specified (or is left blank) WOW will enforce strict behavior in copy, edit, and insert modes.



- **strict message** – This is the error message WOW will display when the user enters a value not among the possible values when strict behavior is being enforced. The default is "You must select a value from the drop down list".

## Auto Complete Advanced Configuration

This section contains information on additional ways to configure and customize auto complete fields.

### Formatted Display Value

In the previous example, we saw how an auto complete field handles display values differently than internal values. There is an optional third type of value, a “formatted display value” which can be used to display additional information in an auto complete drop down. For example, if the user is searching for accounts by state, we can use the drop down to show how many account are in each state, like this:

State =	Massachusetts	
	Maine	9 accts
	Maryland	10 accts
	Massachusetts	2 accts
	Michigan	0 accts
	Minnesota	13 accts
	Mississippi	6 accts
	Missouri	2 accts
	Montana	0 accts

In this situation, each possible value contains 3 distinct values: the internal value (“MA”) which is used internally in database queries but never displayed; the display value (“Massachusetts”) which is shown in the search field; and the formatted display value (“Massachusetts 2 accts”) which is shown in the drop down.

There are two ways to use a formatted display value in the auto complete drop down. The first is to change your possible values query to return a third column of results. The third column should include any HTML formatting, and will be used as the display value. Here is the possible values query which produces the above auto complete drop down.

Basic		
Label	AC State PV - SQL format	Title
Operation Type *	Possible Values	Description
Operation Code	<pre> SELECT STCODE, STNAME, '&amp;'    'nbsp;&lt;SPAN style="position: absolute; left: 0;"&gt;'    STNAME    '&lt;/SPAN&gt;&lt;SPAN STYLE="position: absolute;'    'right: 0; text-align: right; font-size: small; color: olive; padding-top:'    '4px; vertical-align: bottom;"&gt;'    CASE WHEN CNT IS NOT NULL THEN CAST(CNT AS CHAR(2)) ELSE '0' END    ' accts&lt;/SPAN&gt;'  AS CNT FROM JETEMP.STATES LEFT OUTER JOIN ( select distinct cstate, count(*) as CNT from jetemp.customer1 group by CSTATE ) T1 ON CSTATE = STCODE WHERE  STNAME like ??CSTATE or cast(??1 as CHAR(10)) IS NULL  order by STNAME </pre>	

The third column in the query contains both data (the CNT value from the join table) as well as HTML formatting for displaying the data.

The second way to use a formatted display value involves creating a custom Row subclass in Java. If your custom Row subclass is used as the row class by the possible values query, then it should extend the `planetj.dataengine.possiblevalues.AutoCompleteResultRow` class. You can then override the `getFormattedDropDownValue(ExecutionContext)` method to supply the formatted display value for that result row.

When using a formatted display value, it is important to remember that the formatted display value is never shown in the auto complete field itself, only in the drop down. Therefore it is the display value (and not the formatted display value) that is plugged into the possible values query and determines which possible values should be displayed in the drop down.

## SQL-based Auto Complete [PRO]

In the previous examples, once all of the possible values were selected from the database Java was used to filter out those not matching the value entered by the user. In most cases this is adequate, however in cases where there are a large number of values to be filtered, doing the filtering in Java may be a performance issue. It is possible to use SQL to select only values matching the user's entry from the database; this can improve performance but increases the complexity of the possible values SQL.

### Using SQL-based Auto Complete

In order to use SQL-based filtering, you need to set the *type* property in the Auto Complete property group to SQL:

```
AutoComplete{
  type: SQL;
}
```

This informs WOW that the SQL in the possible values operation will take the value entered by the user into account, and therefore it is not necessary to filter the rows returned from the database.

The main query will be exactly the same for both SQL-based and Java-based auto complete. We will use the query from the first example in this section, where the user was searching for county names by entering a state. The SQL for that query was

```
SELECT * FROM jetemp.COUNTY WHERE CNTSTATE = ?
```

In the Java-based auto complete scenario, the SQL for the possible values query was:

```
SELECT STCODE, STNAME FROM JETEMP.STATES
```

```
order by STNAME
```

For an SQL-based auto complete scenario, we need to change the possible values SQL to be:

```
SELECT STCODE, STNAME FROM JETEMP.STATES
```

```
WHERE STNAME like ??CNTSTATE or cast(??1 as CHAR(10)) IS NULL
```

```
order by STNAME
```

This possible values operation will search a file containing both the state abbreviations and the full state names, based on a partial state name entered by the user. The partial state name will be pulled from the CNTSTATE field, which is the field that will be displayed on the screen by the main search query. Including the user's value in the SQL means that all rows returned by this query can be shown to the user in the auto complete drop down – no additional filtering in Java is required. All possible values operations for SQL-based auto complete fields should use a LIKE comparison in the SQL, since the goal is to find all matches which begin with (or contain) a value entered by the user.

## Derived Fields

If you are using SQL-based auto complete, then the possible values SQL query will use the LIKE comparison, which means that only a String based field can be used as the auto complete field. In order to use SQL-based auto complete on a non-character column in the database a derived field is required. (A derived field is a logical field within WOW which does not directly correspond to a database field.) A derived field can also be used in cases where you want to use auto complete on a field while searching, but not when editing that field in a row.

In order to demonstrate using a derived field with auto complete, we will consider a case where we want to search for an account by account number. The account number is stored as DECIMAL data in the database, so we will need to use a derived field in order for auto complete to work. We will display the account number and the name of the account owner in the auto complete drop down.

The first step is to create a derived field descriptor in the table we are querying. You can name this field descriptor whatever you want, however you should make a note of both its name and its ID. The database type and type name should both be set to CHAR – this will cause the field to be created as String field. Ensure that the size of this field is adequate to hold whatever values may be displayed/entered by the user in the search field. (In our case we need to make the field big enough to hold the account number plus the name of the

account owner.)

**Basic Settings**

Field Name\*: ID\_AUTO\_COMPLETE External Name: ID  
Required: ☐ Required On Search: ☐  
Default Value: ☐ -- None -- ☐   
Auto Update Value:

**Advanced Settings**

Field Class: ☒ Enter Class Name:  ☐ Select Existing: Address 1  
Field Descriptor Type\*: Derived Formatter Class:   
Concurrency\*: Concurrent Updates and Deletes Allowed Getter Method:   
Remarks:   
Association Operation: -- None -- Setter Method:   
Notify Status Change: No XML Tag:

**Additional Settings**

Sortable: ☒ Auto Increment: ☐  
Read Only: ☐ Currency: ☐  
Id: 711628 Usage Id:

**Database Settings**

Library Name\*: JETEMP Table Name\*: CUSTOMER  
System Alias\*: DATETIME  
SQL Type\*: CHAR SQL Type Name\*: CHAR  
Column Size: 15 Scale:   
Nullable: Allow Null Key Position:

For the main query, we will want the prompting to use the derived field (which is a String field) as opposed to the normal field for that database column (which is not a String field). This is accomplished by using the ID of the field descriptor in the operation query. In this case, the field descriptor's ID is 711628. We will also have to use the SQL CAST function in order to compare the character data in the derived field to the non-character data in the database field.

Basic		
Label	<input type="text" value="Non-Char"/>	Title
Operation Type *	<input type="text" value="SQL"/>	Description
<pre>SELECT ID, NAME, BALANCE FROM JETEMP.CUSTOMER WHERE CAST(ID AS CHAR(4)) = ?711628</pre>		

Next, we create the possible values operation as usual for the auto complete field. The possible values operation should refer to the value in the derived field, since that is the field displayed on the screen where the user will be entering values into. In our example, our derived field is the ID\_AUTO\_COMPLETE field. This operation will probably also have to use the CAST function to convert the non-character data in the database to CHAR data.

Basic		
Label	<input type="text" value="AC ID PV - SQL formatted"/>	Title
Operation Type *	<input type="text" value="Possible Values"/>	Description
<pre>SELECT ID, ID AS ID2, '&lt;strong&gt;'    CAST(ID AS CHAR(4))    '&lt;/strong&gt; - &lt;span style="color: navy;"&gt;'    NAME    '&lt;/span&gt;' as FMT FROM JETEMP.CUSTOMER WHERE RTRIM(cast (ID as CHAR(4))) LIKE ??ID_AUTO_COMPLETE OR ??1 = " OR CAST(??1 AS CHAR(4)) IS NULL order by ID</pre>		

Finally, return to the derived field descriptor created in the first step, and set its display component to Auto Complete, and its possible values operation to the possible values operation created earlier. (The possible values operation did not exist when we first created the derived field descriptor, or else we would have set it then.)

Basic Settings	
Field Name *	ID_AUTO_COMPLETE
Required:	<input type="checkbox"/>
Default Value:	<input checked="" type="radio"/> -- None -- <input type="radio"/>
External Name:	ID
Required On Search:	<input type="checkbox"/>
Auto Update Value:	

Display Settings	
Field Set:	<input checked="" type="radio"/> <input type="radio"/>
Display Rule *	Always
Help Text:	
Display Width:	
Display Order:	0
Display Component *	Auto Complete
Style Class:	
Display Height:	

Possible Value Settings	
Possible Values Key:	-- None --
Possible Values Operation:	AC ID PV - SQL formatted
Possible Value Class:	<input checked="" type="radio"/> Enter Class Name: <input type="radio"/> Select Existing Class: *DISTINCT*

Now our auto complete search field is set up. As the user types in an account number, the matching account numbers along with the account owner's name is displayed in the drop down.

ID =

15
1 - Amy
10 - Jessica
11 - Kim
12 - Larry
13 - Melvin
14 - Nynaeve
15 - Ollie
16 - Pavlov
100 - Mandy
101 - Eunice

### Auto Complete Fields in Rows

So far, the examples in this section have focused on using auto complete fields as

parameters in a query. This is the most common scenario where auto complete fields will be used. However, if an auto complete field is selected as the result of a query, then the displayed field will retain its auto complete behavior in the results. If you only want to use auto complete during the query prompting and not after the field is selected, then you can use a derived field with auto complete for the query prompting, in which case the fields in the results will not use auto complete. See the Derived Fields heading above for more information on using derived auto complete fields.

If you do want to have auto complete fields in your results, and you are using SQL-based auto complete, then you may need to make further adjustments to the possible values query in order to show the correct display to the user. To demonstrate this we will look at the very first auto complete example from above. The main query was selecting a list of counties by state.


Here is the SQL:

```
SELECT * FROM jetemp.COUNTY WHERE CNTSTATE = ?
```

The first page of results when we run the query looks like this:

State =

**Counties**

 Previous | Next

▲ County Code ▼	▲ State ▼	▲ County ▼
001	MN	Aitkin
003	MN	Anoka
005	MN	Becker
007	MN	Beltrami
009	MN	Benton
011	MN	Big Stone
013	MN	Blue Earth
015	MN	Brown
017	MN	Carlton
019	MN	Carver

Notice that the state column shows the internal value "MN" instead of the display value "Minnesota". When an SQL-based auto complete field appears in the results, by default WOW will show the internal value. This is not a problem if the display and internal values are the same, but in this case they are different, and we want WOW to show the display value. The possible values SQL for the CNTSTATE field (which is the auto complete field) is:

```
SELECT STCODE, STNAME FROM JETEMP.STATES WHERE
STNAME like ??CNTSTATE or cast(??1 as CHAR(10)) IS NULL
order by STNAME
```

The STCODE column contains the internal values, which in our case are state abbreviations like "MN". The STNAME column contains the display values, like "Minnesota".

In a non-auto complete scenario, WOW uses the possible values operation to convert the



internal value into a display value. However for an SQL-based auto complete field, the possible values operation is used to convert a partial display value (entered by the user) into an internal value. The “normal” non-auto complete possible values query for this field would probably look like this:

```
SELECT STCODE, STNAME FROM JETEMP.STATES
```

order by STNAME

In cases like our example we need the possible values operation to do one of two things, depending on whether or not the field is currently being used for an auto complete lookup, or if the field is just being displayed to the user.

The two possible values queries listed above are the same, except for the WHERE clause. So we can combine the two queries into a single possible values query by using the correct WHERE clause:

```
SELECT STCODE, STNAME FROM JETEMP.STATES WHERE
STNAME like ??CNTSTATE or cast(??1 as CHAR(10)) IS NULL
OR NOT (??*AUTO-COMPLETE)
```

order by STNAME

Notice the special ??\*AUTO-COMPLETE parameter. When the possible values query is being used for auto complete purposes, this value will be true, and therefore the WHERE clause will filter rows for the auto complete query. In other cases, such as retrieving the display value for an internal value, ??\*AUTO-COMPLETE will evaluate to false, and the WHERE clause will not filter out any rows.

By constructing a possible values query which uses the ??\*AUTO-COMPLETE parameter in the WHERE clause, your possible values query can be used for both auto complete and non-auto complete scenarios. After changing the possible values query as described above, rerunning the main query gives these results:

State =

## Counties



[Previous](#) | [Next](#)

▲ County Code ▼	▲ State ▼	▲ County ▼
001	Minnesota	Aitkin
003	Minnesota	Anoka
005	Minnesota	Becker
007	Minnesota	Beltrami
009	Minnesota	Benton
011	Minnesota	Big Stone
013	Minnesota	Blue Earth
015	Minnesota	Brown
017	Minnesota	Carlton
019	Minnesota	Carver

# Replacement Libraries

## What is Replacement Library Support

Replacement libraries can be very beneficial when you have multiple libraries that contain the same tables with similar sets of data. A primary example would be test data versus production data. Sometimes, different users have their own libraries, all containing the same files (tables). When a SQL is run, WOW checks to see if there has been a replacement library specified for the library the SQL is about to be run against. If so, the original library in the SQL is switched with a new replacement library.

### For Example:

Let's say there is a replacement library defined to replace LIBRARY1 with TESTLIBRARY1. In addition, we have an SQL statement set to query LIBRARY1 (**SELECT \* FROM LIBRARY1.TABLE1**). With the replacement library defined, the final SQL that is run will actually be **SELECT \* FROM TESTLIBRARY1.TABLE1**. Underneath the covers WOW switches out replacement libraries before executing the SQL

# Four Ways to Implement Replacement Library Support

## WOW Based

WOW based - these replacement libraries take affect for any SQL that is run within the current running WOW instance.

To configure, add a servlet initialization parameter called PJ\_REPLACEMENT\_LIBRARIES (similar to all the other WOW initialization parameters). Servlets that can be accessed by the public are defined in the web application's web.xml file. In this file, you can also define initialization parameters for the servlet. These are parameters that may be used by the servlet when it is initialized. The format for each key value pair is <library to be replaced> = <replacement library>, etc.

### For Example:

```
<init-param id="WOW_Replacement_Libraries">
<param-name>PJ_REPLACEMENT_LIBRARIES</param-name>
<param-value>LIBRARY1=REPLACEMENTLIBRARY32, LIBRARY3=REPLACEMENTLIBRARY2</
param-value>
</init-param>
```

## Application Based

Application based - these replacement libraries take affect for any SQL that is run within the current application.

To configure, add a "Config" property group to the application's properties. Edit the application. Add the Config property group in the properties text area. The format for the library replacement sting value is that same as the other library replacement support implementations <library to replace>=<replacement library>, etc.

### For Example:

```
Config { replacement libraries:
LIBRARY1=TESTLIBRARY4, LIBRARY2=REPLACEMENTLIBRARY31; }
```

## User Based

User based - these replacement libraries take affect for any SQL that is run by current signed in user (for any application the user signs into).

User based replacement libraries take a little more work to configure but can be very useful when dealing with multiple users who have different libraries with similar tables. There are just a couple steps needed to configure user replacement libraries.

1. The SQLOperation used to sign-on the application should contain a column that is to be used for replacement libraries. The format for these column values should be the same format as other library replacement support implementations <library to replace>=<replacement library>, etc.
2. The field descriptor for the "replacement libraries" column from the sign-on needs to have a usage ID set to denote that it is a replacement library field. The usage ID to

denote a replacement library field is -165.

### **For Example:**

Let's say that we have a users file containing the user ID, password, and replacement libraries column. In this file there are two records with the following values:

#### **Record 1**

User Id: USER1

Password: PASSWORD

Replacement Libraries: LIBRARY1=LIBRARY4

#### **Record 2**

User Id: USER2

Password: PASSWORD

Replacement Libraries: LIBRARY1=LIBRARY3, LIBRARY6=REPLACEMENTLIBRARY2

The field descriptor for "replacement libraries" in this file is set with a usage id of -165.

When USER1 signs in and runs any SQL against LIBRARY1, the actual SQL is run against LIBRARY4. On the other hand, if USER2 signs into the same application, any SQL they run against LIBRARY1 will be actually against LIBRARY3

## **URL Based**

URL based - these replacement libraries take affect on any SQL that is run for the current user's environment only. Once the browser window is closed, the replacement libraries are no longer used.

To configure, add a request parameter on the URL call to the application when initially starting the application. This will set and remember the specified replacement libraries for the duration of the use of the application within the current browser session. The format for the library replacement string value is the same as the other library replacement support implementations <library to replace>=<replacement library>, etc. The parameter name for the URL call is '\_pj\_replace\_libs' see below for an example.

### **For Example:**

The following URL would open application with the ID of 1 and use the specified replacement libraries when running SQLs within that application for any user:

<http://www.planetjavainc.com/wow63/runApp?>

[id=1&\\_pj\\_replace\\_libs=LIBRARY1=TESTLIBRARY3,LIBRARY5=REPLACEMENTLIBRARY1](http://www.planetjavainc.com/wow63/runApp?id=1&_pj_replace_libs=LIBRARY1=TESTLIBRARY3,LIBRARY5=REPLACEMENTLIBRARY1)

## Replacement Library Implementation Precedence

If a replacement library is specified on the URL, it will override any other replacement library setting for that library. If specified as a user property, it will override application and WOW global replacement libraries. And finally, application replacement libraries will override WOW global specified replacement libraries.

### For Example:

Let's say the application was run with a URL parameter `_pj_replace_libs=LIBRARY1=REPLACEMENTLIBRARY2`. In addition, we'll say that there is a replacement library specified on the application being run. (`Config {replacement libraries: LIBRARY1=REPLACEMENTLIBRARY4 ;}`).

When an SQL is run, REPLACEMENTTABLE2 would be used because URL replacement libraries take precedence over application replacement libraries.

**[Please click here, WOW Builders Guide continues.](#)**