Web Application Resource for Accessing Relational Entities

Users / Developers Guide

Andre Michel Gauthier, Gauthier Services Limited
<nomad@gserv.co.uk>

Revision History

Revision 0.5	24 April 2003
	Initial Draft

This document provides a straightforward users guide for developing and maintaining applications using WARFARE. There is also some information relating to extending the framework to support access to alternate data sources and to enable developers to extend the basic functionality of the system.

Table of Contents

Overview

WARFARE loosely follows a Model View Controller architecture with a simplified template language providing the display component a View Manager coordinating the transitions between views and a number of View Objects providing data access.

The templates can be utilised in both a push and a pull mode where a template can be called which will pull various views which in turn push their output through other templates for inclusion in the page. Alternatively a simplified ViewRetrieval servlet can be called which will just call one view which will just push output to an appropriate template.

Concepts

A View is defined as a single operation on a data source or a method to request information from a user. In the basic operation of WARFARE there are three types of view, an input view, and update view and a results view. Standard extensions have been added to edit a file on the server and to save a file on the server. Views tend to be owned by a single group and only members of that group will have visibility of that view.

A View is associated with a layout which will define the section template to display the data within and optionally a record template for each record in a results set, or a record template for the user to input data to. A server template is also specified for each layout where error codes from updates or queries can be be displayed to the user or other information if an action fails.

A VHTML pull template is a simple HTML based template that can be used to pull other views into a single page for a user to interact with. This is commonly used to enable groups of related views to be presented to the user in one go in a manner similar to a portal server.

Views are linked together in three main ways, normal links, in-line links and chained links. A normal link will allow information submitted or altered on one page to be submitted via a form post to another view to either run a query or update the data source. This is how an input view can be used to obtain information and pass it into an update view to create a new record or pass data into a results view to search the database and display the results. Results views can also be linked in this way to allow the retrieved information to be edited and passed into an update view to either update the data source or delete the record. A chain link can be used to pass the same information to more than one update view one after the other or from an update view into a results view to see the results of the update or insert into the data source. An in-line link can be used between results views to find information related to the current record from another view and include it in the current page section.

Security and Users

The security model is based around users and groups with each view belonging to a group and only users who are members of that group having access to that view. A user also belongs to a group which is that users primary group. A users user id and primary group id are passed with every query into the system as VMUID and VMGID which can be utilised if required to restrict access to data belonging only to that user or that users group.

Administrating Users

Creating Users

Users can be created from the useradmin.vhtml page in the deployed application by simply typing the user name in the username field in the Manage Users section, selecting the primary group for the user and clicking on the InsertUser button. This will take you to the User Detail Screen

The User Detail Screen

On the UserDetail screen you can add extra details for the user, change the primary group and set their password. You can also at this stage add additional group memberships for this user by clicking the AddMember button.

Group Memberships

When a user is a member of additional groups these will then be displayed on the User Detail screen. The groups to which a user belongs can be changed and updated by clicking the UpdateMembership button or deleted by clicking on the DeleteMembership button.

Searching for users

You can also search for users by entering in the username or partial username with % characters for wildcards in the username field in the Manage Users section and clicking the ShowUsers button. You can restrict the search to specific groups using the pull down box, or by selecting NONE search all groups for matching users. This will display a list of all matching users with a UserDetail button which will link to the User Detail Screen for that user.

Administrating Groups

Creating Groups

Groups can be created by entering a group name in the groupname field in the Manage Groups section field and clicking the InsertGroup button.

The Show Groups Screen

The show groups screen allows you to can edit the group name, list all members of a group or delete the group.

Searching for Groups

You can also search for groups by entering a partial groupname into the groupname field in the Manage Groups section with % characters for wildcards and clicking the ShowGroups button. This will link you straight to the Show Groups screen.

Configuring Database Connections

There are two ways of configuring a database connection to be used by WARFARE. Container managed connection pools and Application managed connection pools. Container managed pools should be used in preference where available as most servlet containers have more flexible configuration available than has been implemented for the application managed pools. Application managed pools however can be used in virtually all containers without needing to alter the configuration of the server to create completely self contained web applications.

Configuring Container Managed Pools

You should first configure a DataSource in your application server. The method of doing this is beyond the scope of this document but should be covered by the documentation for your application server. The minimum requirement is that the DataSource is made available to the WARFARE web application via the JNDI tree. This will nearly always involve installing the jar file containing the driver for the database you are connecting to into your application server. This may involve adding to the CLASSPATH in the startup script but most application severs have a better way of installing external libraries such as placing the jar files in to an ext directory or common/lib directory as in the case of Tomcat.

The web-application now needs to be configured to see the JNDI data source. The first place to configure this is in the web.xml file in the WEB-INF directory of the web application where you will need to add the resource references to the JNDI data source.

Example 1. Resource references in web.xml

```
<resource-ref>
<description>
  Link up to the jdbc/pm data source
</description>
  <res-ref-name>
    jdbc/pm
  </res-ref-name>
  <res-type>
    javax.sql.DataSource
  </res-type>
  <res-auth>
    Container
  </res-auth>
</resource-ref>
```

Finally the View Manager configuration needs to be made aware of the JNDI data source. This is done in the viewmanager.properties file in the WEB-INF/classes directory of the web application. The connection name that is used to reference this data source in the views configuration is appended by a .log to specify a log file for this data source and by a .driver used to specify that this is a JNDI DataSource.

Example 2. Adding a JNDI connection to the viewmanager.properties

```
pm.log=/var/tmp/pm.log
pm.driver=JNDI
```

Configuring Application Managed Pools

Application managed pools are orders of magnitude easier to set up which indicates how much simpler and less resilient they are but if you have no choice then its better than nothing. They are entirely configured within the viewmanager.properties file in the WEB-INF/classes directory of the web application. You will need to specify the jdbc driver name, the URL to connect to the data base, username, password, a log file and the maximum size to allow the pool to grow to.

Example 3. Configuring an application managed DataSource in viewmanager.properties

```
pm.driver=org.postgresql.Driver
pm.url=jdbc:postgresql://localhost:5432/pm
pm.username=nomad
pm.password=
pm.log=/var/tmp/pm.log
pm.maxpoolsize=10
```

You will also need to place the jar file containing the relevant driver into the WEB-INF/lib directory of the web application if you are planning on creating a completely self contained web application. Otherwise the driver will need to be added to the CLASSPATH of the server as with container managed connections.

Administrating The View Manager

Creation and administration of Views and view components is performed via the admin.vhmtl screen. There are three main sections, The Manage Views, the Manage Menus and Manage Layouts. The Manage Views section in the center of the page is for searching for views and creating new views. The Manage Menus section is for searching for and creating menus and finally the Manage Layouts section is used for searching for and creating Layouts.

Administrating Views

Creating Views

A view can be created by typing a new view name into the View Name field, selecting the group that the view should belong to and clicking the InsertView button. This will then take you to the View Detail screen.

Searching for Views

If you wish to maintain an existing view then you can enter a view name in the View Name field (using % characters as wildcards) and select the group that the view belongs to (or NONE if you don't know) and clicking on the ShowViews button. This will list all Views matching your search criteria. If you click the ShowViewDetail button next to the view that you are interested in you will be taken to the View Detail screen where you can maintain the view.

The View Detail Screen

The View Detail screen presents you with a form containing all of the details of the view you are currently maintaining. You can alter the view name which is a short name used mainly to label buttons that link to this view. The long name is made available to the push templates and is used as a more informative descriptive title when displaying the view.

The pull down list of View Types enables you to configure what type of view this is and how it will be processed. The text field labeled sqltmpl should reference a filename relative to the WEB-INF/classes directory of your installed webapp or another location on the CLASSPATH. This file should contain SQL to be executed against your database A convenience method for loading this file into a form for you to edit is available if you click EditSql.

The getidtmpl also references a file in the same way and can be edited using the EditGetIDTmpl. This file is usually used to contain SQL to obtain the primary key of a newly inserted record. The saveidas field defines a label that the results from executing the getidtmpl script should be stored under to pass on to subsequent views.

The connection field defines which database connection should be used to execute any sql against for this view. You can then also select a layout from the Layout pull down menu and alter the group that owns this with the Group Name pull down menu.

Administrating Fields

Fields can be configured from the View Detail Screen. Most view types do not need to have fields configured as the framework will by default render fields from a result set as a simple editable text field. If you wish to override this default (i.e. to display a field as read only) you should configure a field. Input views will always require fields to be configured to make use of the standard layouts as there is no results set for the framework to determine what to display.

Clicking on the ShowFields button on the View Details screen will list all of the fields that have been configured for this view. Clicking on the InsertField button will create a new field for you to configure for this view and then display the list of all Fields. The Show Fields screen allows you to alter the name of the field that this configuration applies to and the Label that should be displayed alongside the field. You can also configure the number of lines you want the text box to be (-1 creates the text box as a password field), the width of the text box and also whether the results should be displayed read-only or hidden. You can also pick a pull down menu to be used when displaying this field.

Once you have configured a field appropriately you should click UpdateField to save the configuration. If you no longer require a field to be configured then you can click the DeleteField button to remove that field from the view configuration.

Administrating Links

Links are administrated in the same way as Fields,clicking on the ShowLinks button will show any links for this view. Clicking InsertLink will create a new link and then display the list of all links. The Show Links screen allows you to select from a pull down list the view that this link will link to. The sequence of the view this is primarily used to determine the order of buttons linking views when displayed on screen. Most importantly you can select from a pull down list which type of link you would like this link to be.

A Normal Link will be displayed as a button on screen when displaying this view. When clicked the button will submit the entire contents of this view as input variables to the view that is linked to. This enables you to find related records in another view i.e. to obtain a list of all invoices for a single client when you are looking at a clients record.

An In-Line link performs in a very similar way in that the details from one view are fed as input parameters to the view that is linked to but with an in-line link the results are displayed on the same page instead of rendered

on a separate page after clicking a button.

A chained link is completely different in that generally only the input parameters from the current view are passed on to the next view in the chain. Any results from executing the getidtmpl will also be passed on under the saveidas variable. This will also prevent the current view from being displayed unless it fails to process. Only the view at the end of a chain is displayed. An example of using chained views would be to chain a Display Details View after a view which has inserted a record. The getidtmpl can determine the primary key of the record that has just been inserted and pass this onto a results view to retrieve the inserted record and display it for further editing.

Once you have edited the link you are interested in and you are happy with it you can click on UpdateLink to save the changes that you have made. If you would like to discard this link then you can click on DeleteLink to delete the link.

Administrating Layouts

You can search for existing Layouts by entering the LayoutName in the layoutname field on the admin screen you can use % as a wildcard if you are uncertain of the layouts name. Clicking the ShowLayouts button will list all Layouts that match your search criteria and allow you to edit them. If you wish to create a new layout then enter the new name for your layout in the Layoutname field on the admin screen and click on InsertLayout.

After searching for a layout or inserting a new layout hopefully one or more layouts will be displayed on screen for you to edit. You can alter the name of the layout here and also change the type of layout that you want this layout to be.

A Horizontal table layout will display all fields in a horizontal orientation across the page with labels across the top of the table multiple records will be repeated down the page. A Vertical table layout will display all fields in a vertically oriented table with labels down the left had side of the page, multiple records will be repeated down the page. A template layout will make use of the recordtmpl template to display each record in the results set or provide an input form. The XML layout will generate an XML representation of the page and if a record template is specified it will be used as an XSLT stylesheet.

There are three fields to specify templates for each layout. These fields should refer to a file relative to the WEB-INF/classes directory of the web application or another location on the CLASSPATH. There are convenience links to EditTemplate, EditRecordTmpl and EditServerTmpl to load these templates into a form field for editing. The first template is simply a section template to provide the gross layout for a view i.e. to provide a section heading or a border around the view. The recordtmpl is used to provide a template for each record in a results set or for the input view. The servertmpl is used to provide feedback to the user from the server. This is primarily used to provide confirmation of successful updates to a database but is used if an update fails or a select statement from a results view fails.

Administrating Menus

You can search for existing Menus by entering the Menu Name in the MenuName field on the admin screen you can use % as a wildcard if you are uncertain of the menus name. Clicking the ShowMenu button will list all Menus that match your search criteria and allow you to edit them. If you wish to create a new menu then enter the new name for your menu in the MenuName field on the admin screen and click on InsertMenu.

After searching for a menu or inserting a new menu hopefully one or more menus will be displayed on screen for you to edit. You can alter the name of the menu here, alter the table a menu is generated from and pick which fields from that table should be used to construct the menu. The realvalue field is the value from the table that you will actually want to use in processing your view, typically this is a primary key value that is not particularly human readable. The displayvalue is the field from the table which you will present to the user to enable them to select an appropriate value.

The restriction field can be used in three separate ways. The most obvious is to restrict the menu based on a simple where clause i.e. to only allow uses to select customers from a list where the status is current set the restriction field to status='current'. A less obvious is to restrict the list by some variable from the data passed into this view. An example of this would be to only select contact names where the company id is the same as the companyid you are processing an order for. To do this simply start the restriction field with an equals symbol followed by the field name you wish to restrict on i.e. =companyid (if a restriction of this type is used then the

resrtiction field needs to be a field of the views in which the menu is to be used). A final example is to restrict the list based on the userid or group that the current user belongs to. This can be used to prevent users from selecting items which they don't have permission to select. To do this you should set the restriction field to either VMUID or VMGID to restrict by user or group respectively.

Writing Templates

Templates in their simplest form are simple files containing some specialised tags which will be substituted with values. The specialised tags are differentiated from any other tags that may be present by being enclosed within <% and %>.

The prop tag

The most common tag that is used within all templates is the prop tag. The prop tag takes a number of attributes that determine what is substituted in place of the tag. The first attribute is the name tag which determines which properties value will be used. The prefix attribute specifies a string that will be prefixed to the replaced value. The suffix specifies a string that will be appended to the replaced value.

Example 4. Using the property tag

If the property spam is set to equal 'tin' then <%prop name='spam' prefix='blue ' suffix=' yellow writing'%> will be replaced with 'blue tin yellow writing'.

If the property specified by the name attribute is empty or not set then nothing will be substituted in place of the tag. If you wish something to be substituted in place of the tag when the property is not set then you will need to specify a default attribute which will be used in place of the property value if it is not set.

All templates have some standard properties available to them that can be used. There are some additional properties available only to certain template types.

General Properties

The properties here are available to most templates as they are generated and fed in from the front end.

VMUNAME	The name of the user logged into the system.
VMUID	The numerical user id of the user logged into the system.
VMGID	The numerical group id of the user logged into the system.
VMSID	The session id of the current users session. This can be used for tracking users or shopping carts.
HTML Parameters	Any variable that has been passed in via a POST parameter or a GET parameter will be available.

Page Template Properties

These properties are generally available to page templates in layouts. The page template is used to provide a consistent wrapper for view presentation.

viewname	The primary view that has been requested.
longname	The primary view that has been requested.

viewdata	The actual data for a view.

Server Template Properties

Server templates are used to provide feedback to the user on what a view has performed. In the case of update views they can be used to show how many rows have been updated, the saved values of any inserted records. If any view failed server templates can be used to display any error codes and the SQL that the server attempted to execute.

returnValue	The value returned by the attempt to process the view. Typically this is a number of rows updated by the server or another number representing completed actions. In the FileSave views this is used to display a full status message after an attempt to save a file.
saveidas	The parameter that a returned id for a newly inserted record will be saved as.
insertedValue	The returned id for a newly inserted record.
SQLString	The SQL that the view attempted to execute on the server.
exception	The exception that the view threw during execution

Record Template Properties

Record templates have a whole host of parameters to represent each field that is returned as part of the results set. These properties are accessed for each field by specifying the field followed by a . and the name of the property. You can also position any sub-views using the subview property.

fieldname.value	The raw value for a field.
.hiddenfiel *fieldname*d	The value for a field as a hidden form field.
fieldname.field	The value for a field as a form field which will take on any field settings specified in the view manager including menus.
subview.*viewname*	A complete sub-view identified by the *viewname*. Sub-views are generated based on any In-Line links specified in the view manager.

vhtml Template Properties

Some additional parameters are available to vhtml templates to incorporate various extra views.

view.*viewname*	A complete view identified by the *viewname*. Views are made available as properties via the import tag.
view.main	The main view for a page that is usually specified via the View parameter in any POST data to the page.

The import tag

The import tag simply loads another view into a vhtml template to make it available as a property to the template. It takes one parameter of the viewname to load. This tag is only valid on a vhtml template.

Example 5. Using the import tag

```
<%import viewname="Announcements"%>
```

The import tag also has an optional target attribute that can be used to define the target view property to send the results from any form submissions to. The target can be set to anything and will be used to replace appropriate property in the vhtml template. If the target attribute is to be used with template or XML views then care must be taken to store the value as a hidden form field. The default display routines will handle this automaticaly.

Example 6. Using the import tag with a target

```
<import viewname="Announcements" target="Announcements">
```

This will result in the results of the search to replace the announcements view in the template

The forcelogon tag

The forcelogon tag takes no parameters and simply forces the user to authenticate as a non guest user before processing a vhtml page. This tag is only valid on a vhtml template.

Extending the Framework

You may find that you can't do everything you want to be able to do entirely within the framework provided here. You may want to access data via EJBs or use SOAP or some other arbitrary API. Rather than attempt to cater for all possible methods of accessing data, or displaying data, a simple method for extending the framework to specify new view types has been developed. This prevents the framework from becoming a piece of Bloat-ware that will cost thousands of pounds to deploy, support, maintain and develop.

The model is based on the plug-in principal where you can generate code to extend the View class and package it in a jar file. In order to deploy your plug-in then all you should need to do is place the jar file in the WEB-INF/lib directory or another location on the CLASSPATH. If you are configuring views directly in the database (i.e. not using the view manager) then all you will need to do is set the viewtype to the class name of your plug-in and the framework will be able to use it. If you want your new view type to be available to the view manager then the view type will need to be registered in the view manager to make it available in the pull down list.

Writing your plug-in

Generating a plugin that can retrieve and display data from an alternative source is simply a case of extending the View class and writing your own processView method to retrieve the data and a displayView method to display that data. The example here shows using some of the View properties from the vm_view table to provide information to retrieve a file from the server. This example does not implement a displayView method but instead extends the ViewInput class which implements a standard set of display routines.

Example 7. The ViewFileEdit Plug-In

```
package gserv.vm;
import gserv.util.*;
import java.io.*;
import java.util.*;

public class ViewFileEdit extends ViewInput{
```

```
    public int processView(Properties props)
      throws ViewException {
      String field=viewConf.getProperty("getidtmpl"); ❶
      String filename = props.getProperty(field,"");
      StringBuffer fileData = new StringBuffer();
      try {
        Class cl = Class.forName("gserv.vm.ViewFileEdit");
        BufferedReader br = new BufferedReader(
          new InputStreamReader(cl.getResourceAsStream(filename)));
        String line;
        while ((line=br.readLine())!=null){
          fileData.append(line+'\n');
        }
      } catch (Exception e) {
        fileData.append("Error reading file");
      }
      results=new GenericDTO[1];
      results[0] = new GenericDTO();
      results[0].setField("filename",filename); ❷
      results[0].setField("filedata",fileData.toString());
      return 1;
    }

}
```

❶ The getidtmpl property of the View is being used to determine which field from the post data specifies the file to be edited.

❷ The filename and data from the file are stored as a GenericDTO and this is passed into the standard display routines and will be displayed using whichever layout has been specified for this view.

Providing a plug-in that can perform updates to an alternative source is again a simple matter of extending the view class and creating your own processView and displayView methods. The example here uses some of the View properties to determine the location to save a file to and then the processView method opens a FileWriter and saves the file. Any error messages from the attempt to

Example 8. The ViewFileSave Plug-In

```
package gserv.vm;
import gserv.util.*;
import java.io.*;
import java.sql.*;
import java.util.*;

public class ViewFileSave extends View{

    public int processView(Properties props){
      String filename = props.getProperty("filename","");
      String data = props.getProperty("data");
      String saveto= viewConf.getProperty("saveidas");
❶
      int rcode=1;
      try {
        PrintWriter pw = new PrintWriter(new FileWriter(saveto+'/'+filename));
        pw.println(data);
        pw.close();
        viewConf.put("returnValue","File Saved OK");
❷
      } catch (IOException e){
        viewConf.put("returnValue","Error on File save:" + e.getMessage());
❸
```

```
        rcode=-1;
      }
      return rcode;
    }

    public void displayView(PrintWriter out){
      ConfirmDisplay indisp = new ConfirmDisplay();
❹
      indisp.setDataProps(viewConf);
      indisp.displayPage(out);
    }

}
```

❶ Here the saveidas property is being used to determine which directory to save the file in.
❷ A positive return value from the attempt to save the file are stored in the viewConf properties to be passed on.
❸ A negative return value from the attempt to save the file are stored in the viewConf properties to be passed on.
❹ One of the standard display classes is being called here as any return value has been stored in a standard location.

Registering your plug-in

Registering plug-ins with the View Manager application is a simple matter of inserting a new entry in the vm_lookup table. The new entry should state that it is a viewtype, specify the class name of the view and assign a name to refer to the view type in the View Manager application.

Example 9. Registering the ViewFileEdit plug-in

```
insert into vm_lookup (lookupid,menuname,value,name) values
(nextval('vm_lookup_seq'),'viewtype','gserv.vm.ViewFileEdit','Edit File');
```

Example 10. Registering the ViewFileSave plug-in

```
insert into vm_lookup (lookupid,menuname,value,name) values
(nextval('vm_lookup_seq'),'viewtype','gserv.vm.ViewFileSave','Save File');
```

Using your plug-in

Once your plugin is registered within the View Manager Application you will be able to select it from the pull down list for viewtype when you are creating or editing views. Obviously your new class will have to be locatable on the CLASSPATH, within a jar file in WEB-INF/lib or locatable within WEB-INF/classes.

You can also use your new view by specifying the class name directly in the SQL used to insert a view into the ViewManager database if you are developing applications directly without using the view manager. All that is required is that the full class name is specified in the viewtype field in the vm_view table. If you don't register the view as above then if a user modifies the view in the view manager then the view type will be set to null or one of the other registered view types.

Example 11. Using the ViewFileEdit plug-in directly

```
insert into vm_view
(viewid,viewname,longname,viewtype,
sqltmpl,getidtmpl,saveidas,
connection,layoutid,groupid)
values
(41,'EditSql','Edit SQL Template','gserv.vm.ViewFileEdit',
'','sqltmpl','SaveFile',
'manager',1,1);
```

Example 12. Using the ViewFileSave plug-in directly

```
insert into vm_view
(viewid,viewname,longname,viewtype,
sqltmpl,getidtmpl,saveidas,
connection,layoutid,groupid)
values
(40,'SaveFile','File Saved','gserv.vm.ViewFileSave',
'','','webapps/data/WEB-INF/classes',
'manager',1,1);
```