WEBADE 4.3.0

User's Guide

Client:	BC Provincial Government
Date:	July 24, 2012
Revision:	2.1

Vivid Solutions Inc.
Suite #1A, 2328 Government St.
Victoria, BC V8T 5G5
Phone: (250) 385-6040
Fax: (250) 385-6046
Website: www.vividsolutions.com

Document Change Control

REVISION NUMBER	DATE OF ISSUE	AUTHOR(S)	DESCRIPTION
1	May 16, 2005	Jason Ross	Original draft
1.1	June 2, 2005	Jason Ross	Revised much of the text and layout.
1.2	August 11, 2005	Jason Ross	Added a section on additional user information available to the developer (Section 2.2.1)
1.3	November 4, 2005	Jason Ross	Updated for WebADE 4.1
1.4	February 20, 2006	Jason Ross	Added documentation.for getAuthorizedUser() methods.
1.5	March 23, 2006	Jason Ross	Fixed getConnectionByAction sample code that was syntactically incorrect.
1.6	July 20, 2006	Jason Ross	Updated the user searching documentation to reflect the newer API, and removed references in the documentation to the deprecated User object.
1.7	September 25, 2006	Jason Ross	Updated the documentation to reflect WebADE 4.1.8 API.
2	March 31, 2008	Jason Ross	Updated document for WebADE 4.2.0
2.1	July 24, 2012	Andrew Wilkinson	Updated document for WebADE 4.3.0

Table of Contents

1. INTRODUCTION TO THE WEBADE 5
1.1 OVERVIEW 5
1.2 PREREQUISITES 6
2. THE WEBADE APPLICATION SINGLETON 7
2.1 TESTING THE SAMPLES OUTSIDE A WEB APPLICATION 7
2.2 USER CREDENTIALS 8
2.3 USER AUTHORIZATIONS 8
2.3.1 SECURED-BY-ORGANIZATION VS NON-SECURED-BY-ORGANIZATION 9
2.3.2 RETRIEVING OTHER USERS' PERMISSIONS 10
2.3.3 OTHER ATTRIBUTES 10
2.4 USER INFORMATION 11
2.4.1 RETRIEVING OTHER USER'S INFORMATION 11
2.4.2 WEBADEUSERINFO ATTRIBUTES 11
2.4.3 ADDITIONAL USER TYPE-SPECIFIC ATTRIBUTES 12
2.4.4 USING THE WEBADEUSERINFO GETATTRIBUTE() METHOD 13
2.4.5 RETRIEVING A LIST OF USERS BY ROLE/ORGANIZATION 14
2.5 RETRIEVING DATABASE CONNECTIONS SECURELY 15
2.5.1 RETRIEVING A DATABASE CONNECTION WITHOUT A USER-CONTEXT 15
2.6 PREFERENCES 16
2.6.1 THE WEBADEPREFERENCES INTERFACE 16
2.6.2 THE WEBADEPREFERENCESET INTERFACE 17
2.6.3 THE WEBADEPREFERENCE INTERFACE 17
2.6.4 THE MULTIVALUEWEBADEPREFERENCE INTERFACE 17
2.6.5 APPLICATION PREFERENCES 18
2.6.6 USER PREFERENCES 18
2.6.7 GLOBAL PREFERENCES 19
3. WEB APPLICATIONS AND MVC DESIGN 20
3.1 WEBADE AND MVC DESIGN 20
3.1.1 USING CUSTOM SERVLETCONTEXTLISTENERS 21
3.1.2 USING CUSTOM FILTERS 21
3.2 STRUTS 22

3.3 STRUTS AND THE WEBADE 22
3.3.1 THE WEBADEACTION CLASS 23
3.4 MORE INFORMATION 23

4. WEB APPLICATION INITIALIZATION 24
4.1 CONFIGURING THE WEBADE 24
4.2 WHAT THE WEBADE DOES AT STARTUP 24
4.3 ADAM 24

5. WEBADE AND MANAGEMENT OF A USER'S SESSION 26
5.1 ORGANIZATION SELECTION 26
5.1.1 CONFIGURING ORGANIZATION SELECTION FOR AN APPLICATION 26
5.2 USER AGREEMENTS 28

6. ADVANCED TOPICS 29
6.1 DATABASE CONNECTIONS AND CONNECTION POOLS 29
6.2 WEBADE EXTENSIONS 29
6.2.1 CREATING A WEBADE EXTENSION 29
6.2.2 REGISTERING A WEBADE EXTENSION 29
6.3 SEARCHING 31
6.3.1 SEARCH OBJECTS AND SEARCH ATTRIBUTES 31
6.3.2 ORGANIZATION SEARCHING 32
6.3.3 USER SEARCHING 33
6.4 MANAGING USER PREFERENCES 33
6.5 ALLOWING BCEID USERS TO VIEW AN IDIR USER EMAIL ADDRESS 35

7. RELATED DOCUMENTATION AND LINKS 36

1. INTRODUCTION TO THE WEBADE

The WebADE is a Java-based J2EE application development framework which aids in delivering common services required by corporate applications. It provides functionality for:

- authorization
- access to user information, such as first and last names and email addresses
- database connection pooling
- logging
- error handling
- pluggable extensions, such as reporting and automated tasks

WebADE provides several advantages to application development, such as presenting simplified APIs for more complex frameworks and providing a common framework for web development that is more tailored to the types of applications that are desired by and for various provincial ministries. Because the WebADE abstracts the actual implementation of these frameworks, there is also the added benefit of being able to upgrade or even swap these underlying frameworks without impacting existing WebADE applications. As the WebADE changes and introduces new functionality, maintaining backwards-compatibility is a high priority, and sometimes even new functionality (such as improved connection pooling or application monitoring) can be introduced into existing applications without requiring any code change at all.

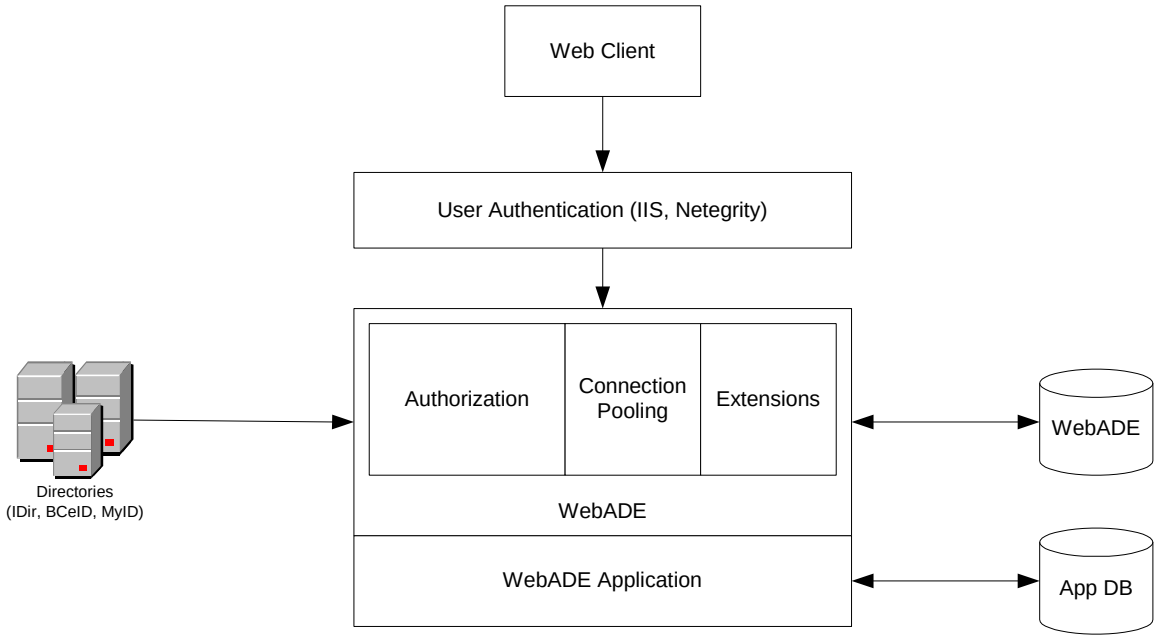
The focus of the WebADE is to provide all of the above benefits to web applications, as these are the most common multi-user applications today, where access to individual components of an application are restricted based on a user's authorizations. However, as most of the core WebADE functionality is not tied to a web environment, it is possible to create a desktop application that uses the WebADE to manage user authorizations. Nevertheless, this document will focus on the integration of the WebADE with web applications, as this is, by far, the primary use of the WebADE.

Finally, it is worthy of note that WebADE 4 introduces several new features, the most important of which is a distributed user authorization management, using a WebADE-integrated application called ADAM.

1.1 OVERVIEW

The WebADE is a central part of an application, aiding in authorizing users for particular application functionality, managing connection pools, and providing a central access point for extensions that can provide additional services, such as reporting or automated tasks.

In a web application, the WebADE sits in the middle, intercepting user requests before they execute business logic, managing database access, restricting a request's database access to that which is granted to the user executing the request, and providing other core operations at both application initialization and the initialization of a user's session.



The WebADE is comprised of several components. From a developer's point of view, there are two main pieces, a set of Java libraries and a supporting database table structure.

The Java libraries use the database to poll for the configuration information of the application, which includes application initialization settings, connection pool configurations, and user authorizations.

1.2 PREREQUISITES

The following are assumed to be in place in preparation for the topics discussed in this guide. Please refer to the WebADE 4 Administrators Guide for more information on any of these subjects:

- WebADE database table structure, including supporting stored procedures, packages, and code table data.
- WebADE application preferences required for internal WebADE use (Such as user provider connection information for IDIR, BCeID and MYID) should be properly set in the WebADE database.
- A database user that has permissions to execute all WebADE stored procedures and packages.
- A WebADE connection jar with the database JDBC URL, user credentials for the above-mentioned user, and any optional connection pool settings that are required for the WebADE connection pool that will be created from these settings.
- A database schema for your application.
- Database users for each WebADE role that requires access to your database.
- The WebADE Java libraries and WebADE connection jar must be included in your application's classpath.

2. THE WEBADE APPLICATION SINGLETON

The main WebADE class is the Application singleton. This singleton contains all methods for user authorization calls, retrieval of connections from the application's connection pools, and provides access to any WebADE extensions configured for your application.

As the core WebADE classes are not dependant on any J2EE code, you can load the WebADE without the need of a web application container, such as OC4J or JRun. This allows you to test the WebADE configuration of your application before deploying your web application. You can also use this to perform unit tests of your business logic using a unit testing API like JUnit.

2.1 TESTING THE SAMPLES OUTSIDE A WEB APPLICATION

All of the examples in this chapter are written assuming they will be run in a web container. As the HttpRequestUtils methods you will see later in this chapter will not work without proper ServletContext and HttpServletRequest instances, you can use the following code examples to create application and user-related objects outside of a web application context for testing purposes.

To create an instance of the Application singleton outside of a web container, run the following code (replacing "APP" with the WebADE application acronym of your WebADE application):

```
import ca.bc.gov.webade.Application;
import ca.bc.gov.webade.WebADEApplicationUtils;

...

Application app = WebADEApplicationUtils.createApplication("APP");
```

NOTE: You **do not** need to use this code inside an actual web application, as this will be created automatically on application initialization.

Once you have the application singleton created, you can create instances of the WebADEPermissions and WebADEUserInfo objects for a target user by writing the following code (replacing "IDIR" and "MYUSER" with the source directory and account name of the user you wish to use in your code):

NOTE: Before WebADE can lookup your test user, you will need to have configured the WebADE to use the appropriate user provider to recognize and connect to the target source directory. See the WebADE 4 Administrator's Guide for instructions on how to do this.

```
import ca.bc.gov.webade.Application;
import ca.bc.gov.webade.WebADEApplicationUtils;
import ca.bc.gov.webade.user.UserCredentials;
import ca.bc.gov.webade.user.WebADEUserInfo;
import ca.bc.gov.webade.user.WebADEUserPermissions;

...
Application app = WebADEApplicationUtils.createApplication("APP");
UserCredentials creds = new UserCredentials();
creds.setSourceDirectory("IDIR");
```

```
creds.setAccountName("MYUSER");
WebADEUserInfo info = app.getWebADEUserInfo(creds);
WebADEUserPermissions auths = app.getWebADEUserPermissions(creds);
```

2.2 USER CREDENTIALS

Since the release of the WebADE 4.1, all users are identified by a UserCredentials object. This UserCredentials object contains the following attributes:

ATTRIBUTE METHOD	DESCRIPTION
getUserGuid	The user's 32-character Hex-value GUID.
getAccountName	The user's unique account name.
getSourceDirectory	The source directory containing the user's account record.
getUserTypeCode	The user's WebADE User Type Code (GOV, BUP, UIN, or VIN).

A user's UserCredentials object can be used to lookup the user's WebADE permissions, by calling the Application singleton's getWebADEUserPermissions() method, and user information, by calling the Application singleton's getWebADEUserInfo() method. To load a user, the configured user provider must support the userTypeCode/sourceDirectory of the user. If an attempt is made to load or search for a user and an unsupported userTypeCode/sourceDirectory is specified, a WebADEUserProviderException will be thrown. If the userTypeCode and sourceDirectory are not specified, the user provider will only attempt to look up the types of users it supports.

NOTE: It is not necessary to know all of a user's credentials, in order to locate their permissions or information. It should be sufficient to supply one of the user's GUID or account name (Preferably the user's GUID) and one of the user's source directory or user type code. When the WebADE locates the user's permissions or information, any unspecified credentials attributes will be set by the WebADE. That said, is it best to supply as much of the user's credentials as possible, as it is possible some of these values could have changed since you last acquired them.

2.3 USER AUTHORIZATIONS

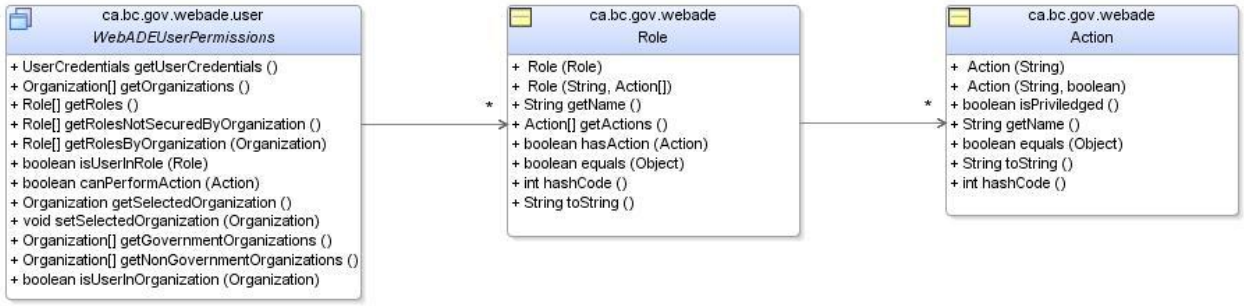
The main purpose of the WebADE is to control a user's access to an application's actions. An action is any self-contained segment of code that can be called by a user. Examples of actions, in plain English, would be "Edit a User's Account" or "View the ABC Report". From a web application perspective, an action is usually (But not necessarily) all of the business processing that results from a user sending a request to the server, such as by clicking on a link or submitting an HTML form.

Each action in an application can be assigned to one or more roles. A role can be viewed as a defined access to an application that contains a collection of actions that are all related from a business perspective. Examples of roles would be "Application User" or "Application Administrator".

By defining these roles in the application, we now have reasonably-sized sections of application code that we can allocate to users, authorizing these users access to the application in the assigned role.

On top of simply granting a role to a user, it is possible in WebADE 4 to grant on behalf of an organization. See the Organization Selection section in this document and the ADAM User's Guide for more information about organization-based role authorization.

Here is a diagram illustrating the WebADE's view of authorizations.



WebADE uses these authorizations to restrict user access at two points: 1) At the point of request, before a business code is executed and 2) At any time that access is required to the application's database.

When a user submits a request, but before any business logic is performed, WebADE should be called to verify that the user can perform the WebADE action associated with that logic. This can be done with code similar to the following (replacing "myActionName" with the name of the target action):

```
import ca.bc.gov.webade.Action;
import ca.bc.gov.webade.http.HttpRequestUtils;
import ca.bc.gov.webade.user.WebADECurrentUserPermissions;

...

HttpServletRequest req = ...;

WebADECurrentUserPermissions user = HttpRequestUtils.getCurrentUserPermissions(req);
boolean canPerform = user.canPerformAction(new Action("myActionName"));

if (canPerform) {
    ...
}
```

2.3.1 SECURED-BY-ORGANIZATION VS NON-SECURED-BY-ORGANIZATION

User permissions come in two types, secured-by-organization and non-secured-by-organization. A user's WebADE role-permission is secured-by-organization if, when the associated Authorization Profile is granted in ADAM to the user, it is granted on behalf of an organization. This means the user only is permitted to perform that role in the target WebADE application while acting for that organization. If an application is using secure-by-organization permissions, it is recommended that the application turn on User Organization Selection to allow WebADE to restrict a user's secured-by-organization permissions in a session to only those for the organization the user selects for that session. See the section User Organization Selection for more information.

If a user is granted a WebADE role-permission through an ADAM Authorization Profile that is not secured-by-organization, the user will have access to that role in every web session they have for a WebADE web application. If User Organization Selection is turned on, the user will still have access to this role, no matter what organization they select to work on behalf of for the web session. To get the list of user roles that are non-secured-by-organization, call the getNonSecuredByOrganizationRoles() method user's WebADEUserPermissions object. For example:

```
WebADECurrentUserPermissions perms = HttpRequestUtils.getCurrentUserPermissions(req);
Role[] userRoles = perms.getRolesNotSecuredByOrganization();
```

2.3.2 RETRIEVING OTHER USERS' PERMISSIONS

You may also request any WebADE user's permissions by retrieving the Application singleton from the ServletContext, and using code similar to the following:

```
import ca.bc.gov.webade.Application;
import ca.bc.gov.webade.http.HttpRequestUtils;
import ca.bc.gov.webade.user.GUID;
import ca.bc.gov.webade.user.UserCredentials ;
import ca.bc.gov.webade.user.UserTypeCode;
import ca.bc.gov.webade.user.WebADEUserPermissions;

...

ServletContext ctx = ...;

Application app = HttpRequestUtils.getApplication(ctx);
UserCredentials creds = new UserCredentials();
creds.setUserTypeCode(UserTypeCode.GOVERNMENT);
creds.setUserGuid(new GUID("EC98010245675A6E0DAE484BE047C8A3"));
WebADEUserPermissions auths = app.getWebADEUserPermissions(creds);
```

2.3.3 OTHER ATTRIBUTES

The WebADECurrentUserPermissions object returned from the HttpRequestUtils.getCurrentUserPermissions() method in the above example is an extended version of the WebADEUserPermissions object in the example above. These permissions objects contain a certain amount of additional information about the user, if it is available to the WebADE. This information is available through the named methods for the following attributes:

ATTRIBUTE METHOD	DESCRIPTION
getUserCredentials	The user's identifying credentials, including account name, GUID, source directory, and user type code.
getOrganizations	The set of organizations the user has application authorizations given on behalf of for the WebADE application.
getRoles	The WebADE application roles the user is authorized for.
isUserAuthenticated	(Current user only) A flag indicating whether the user has accessed the application through some form of authentication.

IsWebADEUser	(Current user only) A flag indicating whether the current user has been located in the WebADE system. If the user has been authenticated but cannot be located within the WebADE itself, this value will be false.

2.4 USER INFORMATION

Since the release of the WebADE 4.1, user information has been separated from the user's WebADE permissions for a specific application. The current user's information can be retrieved with code similar to the following:

```
import ca.bc.gov.webade.http.HttpRequestUtils;
import ca.bc.gov.webade.user.WebADEUserInfo;

...

HttpServletRequest req = ...;

WebADEUserInfo info = HttpRequestUtils.getCurrentUserInfo(req);
```

2.4.1 RETRIEVING OTHER USER'S INFORMATION

You may also request any user's information by retrieving the Application singleton from the ServletContext, and using code similar to the following:

```
import ca.bc.gov.webade.Application;
import ca.bc.gov.webade.http.HttpRequestUtils;
import ca.bc.gov.webade.user.GUID;
import ca.bc.gov.webade.user.UserCredentials ;
import ca.bc.gov.webade.user.UserTypeCode;
import ca.bc.gov.webade.user.WebADEUserInfo;

...

ServletContext ctx = ...;

Application app = HttpRequestUtils.getApplication(ctx);
UserCredentials creds = new UserCredentials();
creds.setUserTypeCode(UserTypeCode.GOVERNMENT);
creds.setUserGuid(new GUID("EC98010245675A6E0DAE484BE047C8A3"));
WebADEUserInfo info = app.getWebADEUserInfo(creds);
```

NOTE: If a call for a user's information is made to the WebADE during the processing of another user's HTTP request, it is possible that the requesting user does not have the permissions to view the target user's information. Developers should take this into account, checking the returned WebADEUserInfo object's isVisible() flag before continuing processing of the HTTP request. If this flag is false, the requesting user is not permitted to view this user's information, and the developer should handle this condition properly.

2.4.2 WEBADEUSERINFO ATTRIBUTES

The WebADEUserInfo object has the following attributes:

ATTRIBUTE METHOD	DESCRIPTION

getUserCredentials	The user's identifying credentials, including account name, GUID, source directory, and user type code.
getDisplayName	The display name for the user, if available.
getFirstName	The first name of the user, if available.
getLastName	The last name of the user, if available.
getMiddleInitial	The middle initial of the user's name, if available.
getEmailAddress	The email address of the user, if available.
getPhoneNumber	The phone number of the user, if available.
getExpiryDate	The date the user's account will expire.
isVisible	A flag indicating whether the requesting user has the ability to view the user's personal information.

2.4.3 ADDITIONAL USER TYPE-SPECIFIC ATTRIBUTES

In additional to the common set of attributes listed above, each WebADE user type can have additional attributes, specific to that particular user type. To view these user type-specific attributes, you must cast the WebADEUserInfo object to the appropriate subclass. Here is an example:

```
import ca.bc.gov.webade.user.BusinessPartnerUserInfo;
import ca.bc.gov.webade.user.GovernmentUserInfo;
import ca.bc.gov.webade.user.IndividualUserInfo;
import ca.bc.gov.webade.user.WebADEUserInfo;

...

WebADEUserInfo info = ...;

if (info instanceof GovernmentUserInfo) {
    GovernmentUserInfo govUser = (GovernmentUserInfo)info;
} else if (info instanceof BusinessPartnerUserInfo) {
    BusinessPartnerUserInfo busPartner = (BusinessPartnerUserInfo)info;
} else if (info instanceof IndividualUserInfo) {
    IndividualUserInfo individual = (IndividualUserInfo)info;
}
```

GOVERNMENTUSERINFO

GovernmentUserInfo has the following additional attributes:

ATTRIBUTE METHOD	DESCRIPTION
getAccountType	The user's BC Gov Account type.
getEmployeeId	The user's employee Id.
isEmployee	A flag indicating if the user is a government employee.

BUSINESSPARTNERUSERINFO

BusinessPartnerUserInfo has the following additional attributes:

ATTRIBUTE METHOD	DESCRIPTION
getBusinessGUID	The user's associated business' GUID.
getBusinessLegalName	The user's associated business' legal name.
getBusinessActivationCode	The user's associated business' activation code.

INDIVIDUALUSERINFO

IndividualUserInfo does not currently have any additional attributes.

2.4.4 USING THE WEBADEUSERINFO GETATTRIBUTE() METHOD

The WebADEUserInfo object has a generic way of fetching a user's attributes. This generic getAttribute() method allows User Providers to support new attributes without having to wait for a WebADE release to add the getter and setter methods to the WebADEUserInfo interface. This loosens the tight-coupling between WebADE and CAP web service releases.

To obtain a user attribute through this generic interface, call the getAttribute() method, passing in the reserved unique string for the target attribute. The WebADE supports the following standard attributes:

webade.user.credentials
webade.user.display.name
webade.user.last.name
webade.user.first.name
webade.user.middle.initial
webade.user.email.address
webade.user.phone.number
webade.user.expiry.date
webade.user.is.visible

For Government users, WebADE supports the following additional standard attributes:

government.user.account.type
government.user.employee.id

For Business Partner users, WebADE supports the following additional standard attributes:

business.user.business.GUID
business.user.business.legal.name
business.user.business.activation.code

Here are a couple of examples using the getAttribute() method:

```
WebADEUserInfo info = HttpRequestUtils.getCurrentUserInfo(request);
UserCredentials creds = (UserCredentials)info.getAttributeValue(WebADEUserInfo.USER_CREDENTIALS);
String accountType = (String)info.getAttributeValue(GovernmentUserInfo.ACCOUNT TYPE);
Date expiryDate = (Date)info.getAttributeValue(WebADEUserInfo.EXPIRY_DATE);
```

RELATED METHODS

The WebADEUserInfo object has two other new methods related to the getAttribute() method; getAttributeNames() and hasAttribute().

The getAttributeNames() will return the array of attribute names for the complete list of attributes the user object contains. Here is an example using this method:

```
WebADEUserInfo info = HttpRequestUtils.getCurrentUserInfo(request);
String[] names = info.getAttributeNames();
for (int i = 0; i < names.length; i++) {
    String currentName = names[i];
    System.out.println("User attribute '" + currentName + "' = '" +
user.getAttributeValue(currentName) + "'");
}
```

The hasAttribute() method returns whether the user object supports the attribute represented by the attribute name string passed in. Here are a couple of examples using this method:

```
DefaultBusinessPartnerUserInfo user = new DefaultBusinessPartnerUserInfo();

assert(user.hasAttribute(WebADEUserInfo.USER CREDENTIALS));
assert(user.hasAttribute(WebADEUserInfo.DISPLAY_NAME));
assert(user.hasAttribute(WebADEUserInfo.LAST_NAME));
assert(user.hasAttribute(WebADEUserInfo.FIRST_NAME));
assert(user.hasAttribute(WebADEUserInfo.MIDDLE_INITIAL));
assert(user.hasAttribute(WebADEUserInfo.EMAIL_ADDRESS));
assert(user.hasAttribute(WebADEUserInfo.PHONE NUMBER));
assert(user.hasAttribute(WebADEUserInfo.EXPIRY_DATE));
assert(user.hasAttribute(WebADEUserInfo.IS_VISIBLE));
assert(!user.hasAttribute(GovernmentUserInfo.ACCOUNT_TYPE));
assert(!user.hasAttribute(GovernmentUserInfo.EMPLOYEE ID));
assert(user.hasAttribute(BusinessPartnerUserInfo.BUSINESS ACTIVATION CODE));
assert(user.hasAttribute(BusinessPartnerUserInfo.BUSINESS_GUID));
assert(user.hasAttribute(BusinessPartnerUserInfo.BUSINESS_LEGAL_NAME));
```

2.4.5 RETRIEVING A LIST OF USERS BY ROLE/ORGANIZATION

WebADE (since 4.1.4) allows a developer to query for the list of users granted access to the current application for a given Role/Organization combination. If only a role is specified, WebADE returns all Users granted access to that Role regardless of any Organization restriction placed on the authorization. If only an Organization is specified, WebADE returns all users with authorizations to work on behalf of the given organization for the given application, regardless of the Role. Examples are shown below:

```
import ca.bc.gov.webade.Application;
import ca.bc.gov.webade.Role;
import ca.bc.gov.webade.http.HttpRequestUtils;
import ca.bc.gov.webade.user.UserCredentials;

...

HttpServletRequest req = ...;

Application app = HttpRequestUtils.getApplication(req);
Role role = app.getRoles().getRole("myRoleName");
Organization org = app.getOrganizationById(00000);

UserCredentials[] creds;
creds = app.getAuthorizedUsers(role);
creds = app.getAuthorizedUsers(org);
```

```
creds = app.getAuthorizedUsers(role, org);
```

NOTE: The getAuthorizedUsers() methods will throw an IndeterminateAuthorizationsException when the given Role/Organization combination is granted to a rule (Example: "All IDIR Users") or an Active Directory group. This is due to the inability to traverse these rules and groups to determine the set of users within. If this does not affect your application, you can prevent this exception from being thrown by calling the appropriate overloaded method that has an additional Boolean flag called "ignoreIndeterminateAuthorizationsErrors" with a value of true. Alternatively, you can call pre-emptively determine whether the call will throw this exception by calling the Application class method "hasIndeterminateAuthorizations()" with the same Role/Organization combination.

2.5 RETRIEVING DATABASE CONNECTIONS SECURELY

In addition to grouping actions, roles also provide separate connection pools, each with permissions set to perform only the database transactions that are associated with that role. This is a redundant level of security to help prevent a user's request from performing any operation that they do not have authorization for.

When your application's business logic requires a connection to the database, you can use WebADE to retrieve a connection for the associated WebADE action passing in the requesting user. This will ensure that a database connection with the appropriate permissions is retrieved. This can be done with code similar to the following (replacing "myAction" with the name of the target action:

```
import ca.bc.gov.webade.Action;
import ca.bc.gov.webade.Application;
import ca.bc.gov.webade.http.HttpRequestUtils;
import ca.bc.gov.webade.user.WebADEUserPermissions;

...

ServletContext context = ...;
HttpServletRequest req = ...;

Application app = HttpRequestUtils.getApplication(context);
WebADEUserPermissions user = HttpRequestUtils.getCurrentUserPermissions(req);
Connection conn = app.getConnectionByAction(user, new Action("myAction"));
```

2.5.1 RETRIEVING A DATABASE CONNECTION WITHOUT A USER-CONTEXT

There are situations that you may require a database connection, but do not have a user to request for a connection on behalf of. For example:

- On application start-up, during initialization.
- In an automated process, running in the background of an application.

The Application singleton allows for these by a concept called "Priviledged actions". A priviledged action is a specially-marked WebADE action that can be assigned to a role, allowing the developer to obtain that connections from that role's connection pool without passing the WebADE a user as context for the request.

To mark an action as priviledged, the entry in the WebADE ACTION database table must have the PRIVILEGED_IND column value set to "Y". By marking this action as priviledged, the developer can use this action to call the Application singleton's method getConnectionByPriviledgedAction(). WebADE will check to verify that this action is marked as priviledged, and then return a connection from the associated role's connection pool. If the given action is not marked as priviledged in the database, the getConnectionByPriviledgedAction() method will throw an exception.

NOTE: Priviledged actions should be used sparingly and only in the appropriate circumstances, as they bypass a level of WebADE security. You may be asked by the ministry, during migration, why your application is using priviledged actions, and you must be able to provide valid reasons for doing so. Priviledged actions should not normally be used during the process of an HTTP request, where there is a user context that can be used to as context for a connection request.

NOTE: It is preferred that priviledged actions be mapped to WebADE Roles that are only assigned priviledged actions, allowing the associated connection pool's database permissions to be restricted to only those permissions needed for the execution of those priviledged actions.

2.6 PREFERENCES

The WebADE also provides access to WebADE Preferences. Preferences come in 5 types: Global, Application, WebADE, Extension, and User. (For more information about Preferences, including the difference between the 5 types, see the WebADE 4 Administrator's Guide)

A developer can access application, user, and global preferences through the WebADE Application singleton.

NOTE: There are two APIs for preferences in the WebADE library. The Preferences, PreferenceSet, and Preference classes have been deprecated in favour of the WebADEPreferences, WebADEPreferenceSet, and WebADEPreference classes (located in the ca.bc.gov.webade.preferences package). This was done to simplify the preferences API, especially with the addition of the ability to save user preferences. The older getGlobalPreferences(), getApplicationPreferences(), and getUserPreferences() methods have been deprecated and replaced with getWebADEGlobalPreferences(), getWebADEApplicationPreferences(), and getWebADEUserPreferences() methods, which return the new WebADEPreferences object. The older methods will still work, but the Preferences classes are now merely wrappers around the WebADEPreferences classes. Existing applications can continue to use the deprecated methods, but new applications should switch to the new methods where possible.

2.6.1 THE WEBADEPREFERENCES INTERFACE

When fetching the application, user, or global preferences, the application singleton will return a WebADEPreferences object. The WebADEPreferences object contains the complete set of preferences (global, application, or user),

divided by preference sub-type, which are further divided into individual preferences and preference sets.

PREFERENCES VS PREFERENCE SETS

Preferences can be divided into individual preferences and preferences grouped together in a set. A preference in the database belongs in a preference set if the PREFERENCE_SET_NAME column is not null.

All preferences with the same preference sub-type and set name will be added to a WebADEPreferenceSet and stored in the WebADEPreferences object using the preference sub-type as a hash table-type key. To obtain the preference set from the WebADEPreferences object, call the WebADEPreferences class' getPreferenceSet() method, passing in the preference sub-type and preference set name as String objects. This method will return the WebADEPreferenceSet object, or null if no set is defined for the given preference sub-type/preference set name combination.

If a preference is defined without a preference set name, it will be stored individually in the WebADEPreferences object using the preference sub-type as a hash table-type key. To obtain the preference from the WebADEPreferences object, call the WebADEPreferences class' getPreference() method, passing in the preference sub-type and preference name as String objects. This method will return the WebADEPreference object, or null if no preference is defined for the given preference sub-type/preference name combination.

2.6.2 THE WEBADEPREFERENCESET INTERFACE

A preference set is mainly a set of preferences, all of which have the same preference set name. Preference sets are used to group like preferences, such as the configuration preferences for a WebADE User Provider (e.g. the CAP Web Services User Provider). To obtain a preference from the WebADEPreferenceSet object, call the WebADEPreferenceSet class' getPreference() method, passing in the preference name as a String object. This method will return the WebADEPreference object, or null if no preference is defined for the given preference name.

2.6.3 THE WEBADEPREFERENCE INTERFACE

A WebADE preference comprises mostly of a preference name and value. The getPreferenceName() method returns the name of the preference, while the getPreferenceValue() method returns the preference value as a String.

2.6.4 THE MULTIVALUEWEBADEPREFERENCE INTERFACE

The MultiValueWebADEPreference interface is a sub-class of the WebADEPreference interface, and supports multi-value preferences. Multi-value preferences are defined as multiple rows in the PREFERENCE database tables with

the same preference type/sub-type/set name/preference name, but with different preference values. If your application expects a preference to have multiple values, you will need to check the WebADEPreference object that is returned from the WebADEPreferences or WebADEPreferenceSet object using an instanceof class check. For example:

```
WebADEPreference pref = prefs.getPreference("test-sub-type", "test-pref-name");
if (pref instanceof MultiValueWebADEPreference) {
    MultiValueWebADEPreference multiPref = (MultiValueWebADEPreference)pref;
}
```

The MultiValueWebADEPreference object has a getPreferenceValues() that returns the List of preference values for this preference.

2.6.5 APPLICATION PREFERENCES

Application preferences are preferences stored in the WebADE database with a PREFERENCE_TYPE_CODE column value of "APP". Application preferences are application-specific, containing information such as a support contact email address. To retrieve the set of application preferences for your application, use the following code:

```
import ca.bc.gov.webade.Application;
import ca.bc.gov.webade.preferences.WebADEPreferences;
import ca.bc.gov.webade.http.HttpRequestUtils;

...

ServletContext context = ...;
HttpServletRequest req = ...;

Application app = HttpRequestUtils.getApplication(context);
WebADEPreferences preferences = app.getWebADEApplicationPreferences();
```

NOTE: The getApplicationPreferences() method will always reload the preferences from the database. If you are frequently calling this method (One or more times per request), you may want to cache these values in the session context, to prevent excessive calls to the database.

2.6.6 USER PREFERENCES

User preferences are preferences stored in the WebADE database with a PREFERENCE_TYPE_CODE column value of "USR". User preferences are used to store preferences that are user-specific for a WebADE application. To retrieve the set of user preferences for a given user (In this example, the current user), use the following code:

```
import ca.bc.gov.webade.Application;
import ca.bc.gov.webade.preferences.WebADEPreferences;
import ca.bc.gov.webade.http.HttpRequestUtils;
import ca.bc.gov.webade.user.WebADEUserPermissions;

...

ServletContext context = ...;
```

```
HttpServletRequest req = ...;

Application app = HttpRequestUtils.getApplication(context);
WebADEUserPermissions user = HttpRequestUtils.getCurrentUserPermissions(req);
WebADEPreferences preferences = app.getWebADEUserPreferences(user.getUserCredentials());
```

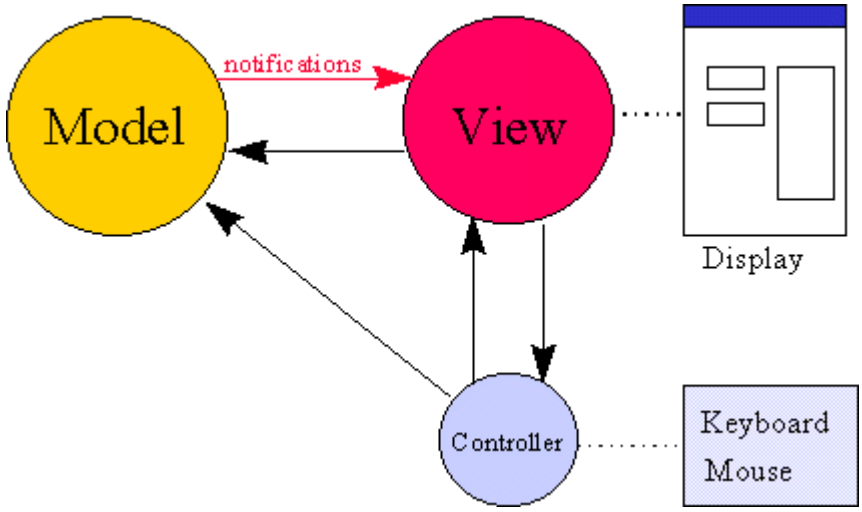
2.6.7 GLOBAL PREFERENCES

Global preferences are preferences stored in the WebADE database with a PREFERENCE_TYPE_CODE column value of "GLB". Global preferences are used to store preferences that are not application-specific, such as a ministry website URL. To retrieve the complete set of all global preferences, use the following code:

```
import ca.bc.gov.webade.Application;
import ca.bc.gov.webade.preferences.WebADEPreferences;
import ca.bc.gov.webade.http.HttpRequestUtils;

...

ServletContext context = ...;
HttpServletRequest req = ...;

Application app = HttpRequestUtils.getApplication(context);
WebADEPreferences preferences = app.getWebADEGlobalPreferences();
```

3. WEB APPLICATIONS AND MVC DESIGN

Web applications developed with WebADE should be developed with the Model-View-Control style of design, or MVC. MVC is considered the industry standard way of designing applications. The purpose of MVC is to separate display logic from business logic, allowing for clean design and ease of maintenance of the application once it has been delivered.



The current suggested framework for MVC for WebADE is Apache Struts, which is described below. Regardless of which framework is used, the main concept of MVC, as it pertains to web applications is that of using a controlling Servlet to handle all requests to a web application. This means that every link and form in web pages that are part of the "View" section of the application will send a request to the "Control" Servlet that controls the web application.

When the "Control" Servlet receives a request, it passes it off to a "Model" component, which then performs the business logic of the action related to the request. How a servlet determines what "Model" component to pass the request off to is all based on information in the request, and is dependant on the framework chosen to build an MVC web application. Again, we describe how Struts implements this in the section below.

3.1 WEBADE AND MVC DESIGN

The best way to load the WebADE is to use a combination of a ServletContextListener to load and initialize the WebADE at application startup, and a Filter to intercept HTTP Requests to allow the WebADE to pre-processing the request, setting the current user's permissions and information in the session and other internal WebADE processes. To do this, you need to add the following code to your WEB-INF/web.xml file (in accordance with the web.xml DTD standards):

```
<context-param>
  <param-name>webade.application.acronym</param-name>
  <param-value>APP</param-value>
</context-param>
```

```
<filter>
  <filter-name>WebADE Filter</filter-name>
  <filter-class>ca.bc.gov.webade.j2ee.WebADEFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>WebADE Filter</filter-name>
  <servlet-name>action</servlet-name>
</filter-mapping>

<listener>
  <listener-class>ca.bc.gov.webade.j2ee.WebADEServletContextListener</listener-class>
</listener>
```

The context-param "webade.application.acronym" is required by the WebADEServletContextListener to load the application singleton for you application. Set the param-value to the application acronym of your application (The same value as the APPLICATION_ACRONYM column value of your entry in the WebADE APPLICATION table).

NOTE: If your are using Struts (See below), ensure your ActionServlet class configuration **DOES NOT** contain a reference to the WebADEActionServlet implemention. The WebADEActionServlet implementation is an older implementation of the same functionality now provided by the ServletContextListener/Filter implementation, and is not needed in this deployment strategy.

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
... More servlet tag config ...
</servlet>
```

3.1.1 USING CUSTOM SERVLETCONTEXTLISTENERS

If you wish to use custom J2EE ServletContextListeners for application logic at startup and you require access to the WebADE application singleton or other WebADE components, you must map your custom ServletContextListener in the web.xml file using a listener tag set, but declared after the one for the WebADEServletContextListener, like the example below.

```
<listener>
  <listener-class>ca.bc.gov.webade.j2ee.WebADEServletContextListener</listener-class>
</listener>

<listener>
  <listener-class>ca.bc.gov.custom.CustomServletContextListener</listener-class>
</listener>
```

3.1.2 USING CUSTOM FILTERS

If you wish to use custom J2EE Filters for application logic and you require access to the requesting user's WebADE permissions and/or information, you can implement the following code in the doFilter() method in your Filter implementation:

```
/**
 * @see javax.servlet.Filter#doFilter(javax.servlet.ServletRequest,
javax.servlet.ServletResponse, javax.servlet.FilterChain)
 */
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
IOException, ServletException {
    WebADEUserInfo user = HttpRequestUtils.getCurrentUserInfo(httpRequest);
    WebADECurrentUserPermissions user = HttpRequestUtils.getCurrentUserPermissions(httpRequest);
        ... more code here ...
 }
```

Also, when adding your filter and mapping to the web.xml file, remember to use the same mapping as the WebADE Filter, but declare your mapping **after** the <filter-mapping> tag for the WebADE Filter (The J2EE spec declares that filters are called in order of appearance of their mappings in the web.xml file).

3.2 STRUTS

The purpose of the Apache Struts project is to provide a framework that implements the MVC model of application design in standard a way that makes application code clear and easy to read. It also adds functionality that makes implementing a web application easier, with classes that handle form submissions, standard tag libraries and a "Tiles" framework that aid in display logic, and an XML file-based configuration that binds an application together, mapping user requests to model code, as well as defining what JSPs a "Model" component will forward the user's web browser to, depending on the result of the business logic operation.

The "Control" Servlet class in Struts is called ActionServlet. The ActionServlet will receive all browser requests and forward these requests off to instances of the Struts class "Action". A developer will sub-class this Action class to perform the desired operation of the application, interpreting the user's request and executing the desired business logic.

NOTE: This should not be confused with WebADE actions. Although the concepts are similar, and it is possible that each Struts Action could map one-to-one with a WebADE action for authorization purposes.

The Struts framework has a lot of nuances, but the intent of this document is not to be a Struts tutorial. For more information, see the Struts project website at: http://struts.apache.org/

3.3 STRUTS AND THE WEBADE

NOTE: While this method of loading the WebADE is still valid, it is recommended that new applications use the ServletContextListener/Filter implementation described above. If you plan on using the WebADEActionServlet to load the WebADE, make sure you **do not** include the WebADEServletContextListener/WebADEFilter references in your web.xml tile, as described above.

WebADE integrates with Struts by extending the ActionServlet class with the ca.bc.gov.webade.http.WebADEActionServlet. This WebADEActionServlet class injects WebADE code into 3 points of the ActionServlet's processing.

1) On initialization of the servlet, WebADE also creates the WebADE database connection pool and loads its configuration settings, WebADE extensions, and application-specific connection pools for use in the web application.
2) Also on initialization, the WebADEActionServlet provides an init() method that can be overridden by the developer to provide application-specific initialization at this time.
3) On receiving a user's request, WebADE will initialize the user's session and check for any organization selection and user agreements that need to be performed before hading the request to the application's request-processing code (in the form of custom Struts Actions.

If you wish to use the WebADEActionServlet to load the WebADE, you are required to extend the WebADEActionServlet class with your own, application-specific implementation and implement the getApplicationCode() method, returning the application acronym of your application (The same value as the APPLICATION_ACRONYM column value of your entry in the WebADE APPLICATION table).

Then, in your web.xml file, you will need to declare your custom servlet as the Struts controller servlet for your application. Below is an example of a proper mapping:

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>com.example.MyWebADEActionServlet</servlet-class>
... More servlet tag config ...
</servlet>
```

3.3.1 THE WEBADEACTION CLASS

Usually, a Struts developer will extend the org.apache.struts.Action class to create their application's actions. However, you may wish to extend the ca.bc.gov.webade.http.WebADEAction class instead. The WebADEAction simply provides a couple helper methods to obtain the ServletContext and Application singleton directly, giving a little streamlining to WebADE development. Extending the WebADEAction class is not necessary for a J2EE application to be WebADE compliant.

3.4 MORE INFORMATION

Please see the Apache Struts website for more information about configuring and using Struts.

4. WEB APPLICATION INITIALIZATION

As any web application has many resources that need to be initialized before the application can handle user requests, such as connection pools, resource files, or even autonomous processes, a certain amount of processing must occur at the point the application is started inside the web container. This section gives details about this initialization, as well as the overall configuration and administration of a WebADE application.

4.1 CONFIGURING THE WEBADE

Most of the WebADE's settings are configured in the WebADE database. This allows for security of configuration settings, restricting access to these settings to database administrators, and a separation of the configuration from the compiled application WAR or EAR file, allowing settings to change in the application without the need to redeploy.

Of course, the primary difficulty with using a database is that WebADE needs to be made aware of how to connect to that database. This is achieved by creating a compiled Java library of a properly configured WebADEConnection class. For information on configuring and creating this library, see the WebADE 4 Administrator's Guide.

4.2 WHAT THE WEBADE DOES AT STARTUP

When the WebADE is loaded at web application initialization by the controlling Servlet, it performs the following initialization steps.

1) WebADE loads the WebADE database connection settings from the WebADEConnection jar, creating a connection pool for this database, to be used internally.
2) WebADE connects to the WebADE database, loading the application's role and action settings.
3) For each role, WebADE creates a connection pool, using the connection pool settings stored in the WebADE database. If a role does not require a WebADE connection pool (No connection pool settings are in the WebADE database for this role), this step is skipped.
4) WebADE loads the LDAP configuration settings for all domains supported by the web application. Connections to LDAP are used to load user information for all users of the application, excluding authorization information.
5) WebADE loads any WebADE extensions configured for the application, storing a reference to them in the Application singleton.

4.3 ADAM

ADAM is the web application that is used to manage user access to WebADE applications. As a WebADE database is used to manage any number of web applications, ADAM is used to manage the authorizations for these users for all applications installed in a given WebADE database.

ADAM provides two main advantages; centralized, consistent method of granting users to applications, regardless or the differences in applications, and a distributed management of these authorizations. ADAM allows top-level administrators to delegate applications the management of user authorizations for a defined set of users, such as all ministry workers or a private company, to another individual. This delegation restricts that individual to only be able to administrate a subset of users, and only for the application roles that the top-level administrator authorizes the individual for.

It should be noted that ADAM is a very complex application, and a much more thorough explanation can be found in the ADAM User's Guide.

5. WEBADE AND MANAGEMENT OF A USER'S SESSION

When a user connects to a WebADE web application, WebADE must manage a certain amount of information about the user and associate it with the user's session. At the creation of a new session, WebADE loads the user's personal information, such as name, user id, and email address and makes this available to the web application for use in any business logic that may require it. Also at session creation, the WebADE performs two optional operations; organization selection and user agreements.

5.1 ORGANIZATION SELECTION

As an option, while configuring your WebADE application's authorizations in ADAM, you can set it up so that when users are granted access to a role, it is done on behalf of an organization. This means that, when the user creates a session with the web application, they do so on behalf of an organization. The primary advantage of doing this is to allow a user to have varying levels of authorization on an organization-by-organization basis.

For example, user John Doe could have "User" access to an application on behalf of company ABC, but he could be granted "Administrator" access to an application on behalf of company XYZ. When John logs into the web application, he will be presented with a list of all organizations that he has been granted access to the application on behalf of. When he selects one of the organizations from the list, he will be restricted in access to the application to only the authorizations granted to him on behalf of the selected organization.

Granting a user authorization on behalf of an organization can be done using ADAM. Please see the ADAM User's Guide for more information.

5.1.1 CONFIGURING ORGANIZATION SELECTION FOR AN APPLICATION

When a WebADE application secures some or all of its roles by organization, it is often desirable to have the WebADE application require users to select one of the organizations that they have been authorized access to the application on behalf of, at the creation of a new web session with the application.

To enable organization selection for a WebADE web application, you will need to add and configure the Organization Selection Filter in your application's web.xml file.

NOTE: When adding the Organization Selection Filter to your application, you should add a filter-mapping for this filter matching each WebADE Filter filter-mapping configured in your application. When adding the filter-mappings make sure that the Organization Selection Filter filter-mapping is added after the matching WebADE Filter filter-mapping, as the Organization Selection Filter expects the WebADE Filter to have already handled the request.

See the example below for a basic configuration of the Organization Selection Filter:

```
<filter>
    <filter-name>WebADE Filter</filter-name>
    <filter-class>ca.bc.gov.webade.j2ee.WebADEFilter</filter-class>
</filter>

<filter>
    <filter-name>Organization Selection Filter</filter-name>
    <filter-class>ca.bc.gov.webade.j2ee.OrganizationSelectionFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>WebADE Filter</filter-name>
    <servlet-name>action</servlet-name>
</filter-mapping>

<filter-mapping>
    <filter-name>Organization Selection Filter</filter-name>
    <servlet-name>action</servlet-name>
</filter-mapping>

<servlet>
    <servlet-name>action</servlet-name>
    <display-name>Struts Action Servlet</display-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
</servlet>
```

If this filter is added to your application, when a user logs in, they will be presented with an organization-selected page, instructing them to select one of the organizations on behalf of which they have application authorizations, to use as their selected organization for the current web session. Once they select an organization, they will only have access to the application functionality specific to the application roles granted to them on behalf of the given organization.

NOTE: If a user checks the "Set as Default" flag on the organization-selected page when choosing an organization, that organization will be highlighted in the organization drop-down list by default any time in the future that a user logs in to the application and is presented the organization-selected page. This is called the user's default organization.

NOTE: If a user only has application roles granted on behalf of one organization, this organization will automatically be set as the selected organization, and they will not be prompted by the organization-selected page.

SELECT-BY-ORGANIZATION-TYPE SETTING

If you want to have the Organization Selection Filter only allow the user to select their session organization from government organizations or only from non-government organizations, add an init-param to the filter declaration with a param-name of "webade.default.organization.select.by.organization.type" and a param-value of "government" for government organization only selection or "non-government" for non-government organization only selection. For example:

```
<filter>
    <filter-name>Organization Selection Filter</filter-name>
    <filter-class>ca.bc.gov.webade.j2ee.OrganizationSelectionFilter</filter-class>
    <init-param>
        <param-name>webade.default.organization.select.by.organization.type</param-name>
        <param-value>non-government</param-value>
```

```
    </init-param>
</filter>
```

USE-DEFAULT-ORGANIZATION SETTING

If you wish to have the user's default organization (see above) selected automatically (thus bypassing the organization-selection page completely) when the user logs in to your application, add an init-param to the filter declaration with a param-name of "webade.use.default.organization.enabled" and a param-value of "true". For example:

```
<filter>
    <filter-name>Organization Selection Filter</filter-name>
    <filter-class>ca.bc.gov.webade.j2ee.OrganizationSelectionFilter</filter-class>
    <init-param>
        <param-name>webade.use.default.organization.enabled</param-name>
        <param-value>false</param-value>
    </init-param>
</filter>
```

CUSTOM DEFAULT-ORGANIZATION-SWITCH-PAGE SETTING

If you wish to use a custom organization-selection page, add an init-param to the filter declaration with a param-name of "webade.default.organization.switch.page" and a param-value set to the path, relative to the root of the application, for the custom organization selection JSP. For example:

```
<filter>
    <filter-name>Organization Selection Filter</filter-name>
    <filter-class>ca.bc.gov.webade.j2ee.OrganizationSelectionFilter</filter-class>
    <init-param>
        <param-name>webade.default.organization.switch.page</param-name>
        <param-value>test.jsp</param-value>
    </init-param>
</filter>
```

5.2 USER AGREEMENTS

Another optional component is the ability to present the user with a user agreement. User agreements are documents that a user must agree to before gaining access to the application. If your application has any agreements set when a user logs in that the user has not agreed to, they will be presented with the agreement and required to agree to it before accessing the application.

ADAM does not provide the ability to assign user agreements to an application. This must be done via preferences at this time. Please see the ADAM Administrator's Guide for more information.

6. ADVANCED TOPICS

The following topics are considered more "advanced", and it is possible for a WebADE application to be developed without using any of this functionality. If this is your first WebADE application, we would recommend you start developing using what you have learned so far, returning to this section once you feel comfortable with the core WebADE API.

6.1 DATABASE CONNECTIONS AND CONNECTION POOLS

WebADE provides a robust connection pool API based on the javax.sql.PooledConnection specification. WebADE connection pools have the advantages of being very configurable, self-monitoring, and provide extensive logging for debugging purposes.

WebADE connection pools will even self-close connections, statements, and result sets that a developer forgets to close before the object leaves scope, providing helpful logging for the developer to help detect such situations so they can be cleaned up in the development process. As issues like these can easily make it into a production environment without notice, these measures greatly reduce administrative problems that arise when an application is deployed.

For more information on WebADE connection pools, please see the WebADE Connection Pooling Guide.

6.2 WEBADE EXTENSIONS

WebADE allows extensions to be created and registered with the core WebADE at runtime. A registered extension has access to the WebADE database connection, as well as a reference to the Application singleton.

6.2.1 CREATING A WEBADE EXTENSION

To create a new WebADE extension, create a new class, extending the ca.bc.gov.webade.WebADEExtension class. Please note that extensions of this base class are intended to be singleton objects, so your code should be developed with this in mind.

Your extension singleton class must have a public default constructor. This constructor is called during the start-up of the web application on the server, when extensions are registered with the application singleton.

6.2.2 REGISTERING A WEBADE EXTENSION

In order to register a WebADE extension with the application singleton, it needs to be configured with a valid set of preferences stored in the WebADE PREFERENCES table. A WebADE extension requires the creation of two types of preferences; extension and application. This section assumes the reader is familiar with WebADE preferences and understands how to create WebADE

preferences of all types. For a detailed explanation of WebADE preferences, please see the WebADE Administrator's Guide.

EXTENSION PREFERENCES

WebADE extension-type preferences are used exclusively to initialize WebADE extensions. A WebADE extension uses these preferences to initialize itself at application start-up.

A WebADE extension can have any number of preferences and preference sets, and naming for these are up to the extension developer. There are only three mandatory rules that must be followed:

1) All extension preferences must have a PREFERENCE_TYPE value of "EXT".
2) All extension preferences for the same extension must share the same "preference sub-type" name. This name should be the extension's name (such as "reporting" or "services") in lowercase letters.
3) There must be a preference defined for the extension with a null preference set name, a preference name of "extension-class-name", and a value of the fully-qualified class name of the class that extends the ca.bc.gov.webade.WebADEExtension abstract class (Example: "ca.bc.gov.webade.myextension.MyExtension").
4) There must be a preference defined for the extension with a null preference set name, a preference name of "enabled", and a value of either "true" or "false". With a value of "true", the extension will be loaded by the WebADE at startup. With a value of "false", the extension will be ignored at startup.

OTHER EXTENSION REGISTRATION NOTES

When an extension is registered with the application class, it is handed a reference to the application object, which can be accessed by the getApplication() method in the WebADEExtension class.

Connections to the WebADE database can be obtained by calling the getADEConnection() method. Please note that connections obtained this way should be closed by calling the releaseADEConnection() method, instead of calling the Connection.close() method.

EXISTING WEBADE EXTENSIONS AND THE NEW PREFERENCES API

For existing WebADE Extensions (Compiled against a WebADE release before version 04_01_08) to be successfully built against the current WebADE API, they will need code change, extending the WebADEExtension abstract method "init(ca.bc.gov.webade.preferences.WebADEPreferences)". This method should be used instead of the now deprecated "init(ca.bc.gov.webade.Preferences)" method to initialize the extension. See the section on the WebADEPreferences API in this document for information on how to use this new API.

NOTE: Existing WebADE Extension binaries **do not** require code change to be used in WebADE applications using the current release of the WebADE. Only new releases of existing WebADE Extensions require this change.

6.3 SEARCHING

It is sometimes necessary for a WebADE application to perform searches of users and organizations. Searching for each type is similar in implementation, but is flexible enough to allow for different data sources within the same WebADE system.

As each WebADE environment is unique, searching for user and organization data within the WebADE can vary slightly from system to system. For instance, in one system, user's email addresses may not be supported, and so searching for them would not produce any results. In another system, it may not be possible to search by first name or last name. Fortunately, WebADE's searching functionality allows for developers to determine at runtime the available searchable attributes for users and organizations, allowing for flexibility without code change.

6.3.1 SEARCH OBJECTS AND SEARCH ATTRIBUTES

Searching for users and organizations both involve populating the appropriate Search Object's Search Attributes, and submitting this search object to WebADE.

SEARCH OBJECTS

A Search Object is a metadata object that defines the searchable attributes for the target data type (WebADEUserInfo or Organization), including the attribute's type, allowable values, supported flag, and other options, like optional indicator and wildcard search settings.

Each attribute of a Search Object will be represented by a class that extends the SearchAttribute abstract class. This class has one property, the "supported" flag. This flag is set for the specific WebADE environment, and indicates whether this attribute can be used for searching in that particular WebADE instance. The subclasses of SearchAttribute are described below.

TEXTSEARCHATTRIBUTE

This attribute class defines a field that is searchable by any text pattern. This means that there is very little validation that can be performed. This searchable field would be presented to the user as an editable text field.

The TextSearchAttribute has two additional properties: searchValue and wildcardOption. The searchValue should be set to whatever value the user typed in the text field. The wildcardOption should be set to one of: WildcardOptions.EXACT_MATCH, WildcardOptions.WILDCARD_LEFT, WildcardOptions.WILDCARD_RIGHT, or WildcardOptions.WILDCARD_BOTH.

- EXACT_MATCH means to search for values that match exactly what the user typed.
- WILDCARD_LEFT means to search for values that end with the value the user typed.

- WILDCARD_RIGHT means to search for values that start with the value the user typed.
- WILDCARD_BOTH means to search for values that contain the value the user typed.

OPTIONSEARCHATTRIBUTE

This attribute defines a field that is searchable only by a fixed range of values. This searchable field would be presented to the user as a dropdown list.

The OptionSearchAttribute has three additional properties: searchValue, searchOptions, and optional flag. The searchValue should be set to whatever value the user selected from the dropdown list. The searchOptions are the values to be used to populate that list, and the optional flag indicates whether the user must select a search value for this attribute.

DATESEARCHATTRIBUTE

This attribute defines a date field that allows a user to search for a specific date. This attribute has three properties: maxStartDate, maxEndDate, and searchDate. The maxStartDate and maxEndDate define the earliest and latest valid search date the user can enter, while the searchDate is to be set to the date the user entered.

6.3.2 ORGANIZATION SEARCHING

Searching for organizations is performed in three steps: retrieving an OrganizationSearchObject instance, populating the search object with details of the search, submitting the search object to WebADE and iterating the search results.

RETRIEVING AN ORGANIZATIONSEARCHOBJECT INSTANCE

To obtain a properly configured OrganizationSearchObject from WebADE, call the ca.bc.gov.webade.Application method "getOrganizationSearchMetadata()". This method returns an OrganizationSearchObject instance that is properly configured for WebADE organization searching within the specific WebADE instance.

POPULATING THE SEARCH OBJECT

Organizations, by default, allow for searching by organization name and type code. The OrganizationSearchObject has two search attributes; name and organizationTypeCode. The name attribute is a TextSearchAttribute, as described above, while the organizationTypeCode, is an OptionSearchAttribute.

SUBMITTING THE QUERY TO WEBADE AND RETRIEVING THE RESULTS

Once you have populated the Search Object with the desired search criteria, pass this Search Object back to the WebADE, by calling the Application's findOrganizations() method. This method will return a List of matching Organization objects, which you can then iterate over.

6.3.3 USER SEARCHING

Searching for users is similar to organization searching, with the exception that, instead of one Search Object, there is one search object for each source directory supported by this WebADE installation. Examples of source directories would be the IDIR and BCeID domains.

RETRIEVING THE USERSEARCHOBJECT INSTANCES

To obtain the set of properly configured UserSearchObject instances from WebADE, call the ca.bc.gov.webade.Application method "getUserSearchMetadata ()". This method returns a List of UserSearchObject instances that is properly configured for WebADE user searching within the specific WebADE instance.

POPULATING THE SEARCH OBJECT

First you must iterate over the list of UserSearchObjects, calling the getSearchDirectory() method and comparing the value with the target domain you wish to seach for users in. The UserSearchObject has many search attributes, including user id, first name, last name, phone number, email, middle initial, and GUID. Not all attributes are supported for all source directories, so you will have to call the isSupported() method at runtime on each attribute to determine if it can be used for your search.

SUBMITTING THE QUERY TO WEBADE AND RETRIEVING THE RESULTS

Once you have populated the Search Object with the desired search criteria, pass this Search Object back to the WebADE, by calling the Application's findWebADEUsers() method. This method will return an array of matching WebADEUserInfo objects, which you can then iterate over.

NOTE: When searching for users in a WebADE database application, users are located by directly querying the user-provider (ex: CAP web-services), and entries in the WebADE database user table are ignored. If a user is not in the user table, the findWebADEUsers() method will not make an entry for them. To ensure that a target user is in the WebADE database user table, you must call the Application's getWebADEUserInfo() method after the user search.

6.4 MANAGING USER PREFERENCES

A WebADE application can manage user preferences (edit, add, and delete) for each user of that application. This allows a developer to customize the application for each user triggered by runtime actions like user-input.

To manage a user's preferences, you must first obtain the user's WebADE User Preferences from the WebADE application singleton (See User Preferences).

EDITING A PREFERENCE

To edit an existing preference, first retrieve the desired preference from the user's WebADEPreferences object. If the preference is in a preference set, use the getWebADEPreferenceSet() method to retrieve the preference set, and then

call the getWebADEPreference() method on this preference set to get the desired WebADEPreference. If the preference is not in a set, retrieve the desired preference from the user's WebADEPreferences object, using the getWebADEPreference() method. After you have the WebADEPreference object, set the preference value to the desired value using the object's setPreferenceValue() method.

ADDING A NON-PREFERENCE-SET PREFERENCE

To add a brand new preference to a user's preferences, first, create a new WebADEPreference using the DefaultWebADEPreference concrete class (also in the ca.bc.gov.webade.preferences package), passing in the preference name as a parameter to the constructor. Before adding the preference to the user's WebADEPreferences instance, you must set the preference value using the setPreferenceValue() method. Then, call the addPreference() method on the WebADEPreferences object, passing in the preference sub-type as a String (all preferences must have a sub-type), and the WebADEPreference object as the new preference.

ADDING A PREFERENCE-SET PREFERENCE

To add a brand new preference to an existing preference set in a user's preferences, fetch the desired preference set from the user's WebADEPreferences object, using the getWebADEPreferenceSet() method, passing in the preference set's preference sub-type and preference set name. Then, create a new WebADEPreference using the DefaultWebADEPreference concrete class (also in the ca.bc.gov.webade.preferences package), passing in the preference name as a parameter to the constructor. Before adding the preference to the preference set, you must set the preference value, using the setPreferenceValue() method. Then, call the addPreference() method on the WebADEPreferenceSet object, passing in the WebADEPreference object as the new preference.

DELETING A NON-PREFERENCE-SET PREFERENCE

To delete a preference that is not in a WebADEPreferenceSet, call the WebADEPreferences instance's removePreference() method, passing in the preference sub-type and preference name as Strings.

DELETING A PREFERENCE-SET PREFERENCE

To delete a preference that is in a WebADEPreferenceSet, call the WebADEPreferenceSet instance's removePreference() method, passing in the preference sub-type and preference name as Strings.

SAVING THE USER'S PREFERENCE

After you have finished modifying a user's preferences, you must save them back to the database, using the WebADE Application singleton method saveWebADEUserPreferences(), passing in the user's credentials and the modified WebADEUserPreferences object. WebADE will save the user's preferences back to the database.

NOTE: You may not edit or add a multi-value user preference using this API. Doing so will result in a WebADEException when trying to save these preferences to the database.

6.5 ALLOWING BCEID USERS TO VIEW AN IDIR USER EMAIL ADDRESS

The CAP web services do not currently allow a BCeID user logged in to a WebADE application to view IDIR users' email address. However, certain applications require this email address for application functionality.

WebADE has a setting that will allow this CAP web services to be overridden, using the email address from the WebADE database and returning it to the user. To turn this functionality on, you will need to add the following WebADE preference to the CAP web services user provider ("bceid-web-services-provider") by adding it in the PREFERENCE table of the WebADE database:

COLUMN NAME	COLUMN VALUE
PREFERENCE_ID	preference_seq.NEXTVAL
PREFERENCE_TYPE_CODE	"WDE"
PREFERENCE_SUB_TYPE	"user-provider"
APPLICATION_ACRONYM	"*your-application-acronym*"
PREFERENCE_SET_NAME	"bceid-web-services-provider"
PREFERENCE_NAME	"load-idir-email-address-for-bceid-users"
PREFERENCE_VALUE	"true"

7. RELATED DOCUMENTATION AND LINKS

WebADE 4 – Administrator's Guide	http://www.webade.org/
WebADE 4 - Connection Pooling Guide	http://www.webade.org/
WebADE 4 – What's New	http://www.webade.org/
Javasoft Website	http://java.sun.com/
Struts Project Website	http://struts.apache.org/
ADAM User's Guide	http://www.webade.org/