CUSTOMIZATION GUIDE

GVC.SITEMAKER

VERSION 4.1.1

OCTOBER, 2006

CHUCK HILL

Project Sponsor: GVC Board of Directors

GVC.SiteMaker Customization Guide

REVISION CHART AND COPYRIGHT NOTICE

Version	Primary Author(s)	Description of Version	Date Completed
Draft	Chuck Hill	First complete draft.	2002-09-17
3.1	Chuck Hill	Added LDAP Customization	2003-10-12
3.1	Chuck Hill	Added sections on binding customization	2003-11-21
3.1	Chuck Hill	Final LDAP Edits	2003-12-09

CONTENTS

Introduction

This document describes the planned customization points in GVC.SiteMaker. There may also be unplanned customization points and more planned customization points are likely to be added in the future as actual customization needs become clearer. This document will not discuss the configuration options for GVC.SiteMaker as these are discussed in the Configuration Instructions.

The configuration points in SiteMaker are varied and it will take some time to become familiar with them all and when to use which. The focus point of all customizations is the GVCSMCustom framework for that installation under the Custom folder in the GVCSiteMaker project. This framework holds custom configuration information, sub-classes, and UI elements unique to that installation.

Custom EO Subclasses

To alter or extend the behavior of an EO object create a sub-class named <EntityName>Custom.java. This class will be automatically detected and used in place of the original <EntityName>.java. It is not mandatory to create such a sub-class for each EO, but any found will be used. By convention these custom sub-classes should go in the EOs sub-project of the GVCSMCustom framework. See the constructor of SMApplication and the SMEOUtils class (both in GVCSMCore) for details on how this works.

Creating an EO sub-class is primarily useful for altering existing behavior, or adding extra methods to be used in bindings in a WOD file.

Special Appserver Subclasses

The GVCSMCustom framework contains sub-classes of the GVCSMCore classes SMApplication, SMDirectAction, and SMSession (which are in turn sub-classes of WOApplication, WODirectAction, and WOSession). The classes are named SMCustomApplication, SMCustomDirectAction, and SMCustomSession. These sub-classes are used in both GVCSiteMaker and GVCSMAdmin and can be altered or extended in GVCSMCustom. Care should be taken, especially with SMCustomDirectAction as changes will affect both GVCSiteMaker and GVCSMAdmin. Separate sub-classes were not created for each as the intention is to eventually merge these two applications.

Page Mapping

Page Mapping is a technique whereby any WOElement sub-class can be automatically replaced by another WOElement sub-class. Page Mapping was originally created to handle simple page substitution, hence the name, but works equally well for smaller components and dynamic elements. In GVC.SiteMaker Page Mapping is used to allow any page or component from GVCSiteMaker, GVCSMAdmin, and GVCSMCore to be replaced by an equivalent in an installation's GVCSMCustom framework. In this manner nearly any UI customization desired can be accomplished. A number of reusable components (see below) have been created in GVCSMCore to support common Page Mapping needs.

GVCSiteMaker and GVCSMAdmin each has its own page map controlled by two files in GVCSMCustom, GVCSiteMakerPageMap.plist and GVCSMAdminPageMap.plist. The syntax is very simple,

```
    <original page name> = <name of page to use>;
```

For information on how Page Mapping works, see these methods in SMApplication in GVCSMCore: initializeApplicationConfiguration(), pageWithName(), pageMap(), pageNameMappedToName().

In the current project organization of GVC.SiteMaker there are a few restrictions on what components and pages can be mapped. As newly created pages are often cast from WOComponent to their more exact sub-class, for example:

```
    RequestPage requestPage = (RequestPage) pageWithName("RequestPage");
    requestPage.setWebsite(website());
```

the replacement class must be a sub-class of the one being replaced so that the JVM does not raise class cast exceptions. The effect of this is that only the components in GVCSMCore are guaranteed to be mappable. For components in GVCSiteMaker and GVCSMAdmin, the code will need to be examined to determine if casting takes place or not. It is likely that in the future all the components in GVCSiteMaker and GVCSMAdmin will be moved to GVCSMCore or a parallel framework to support mapping. This will result in the applications being mostly empty skeletons.

There is a bug in WebObjects (up to version 5.1.3, possibly corrected in next version), which causes Page Mapping to not work for WOComponent sub-classes that are not top-level pages. The work around for this is to use them in a WOSwitchComponent like this:

```
    SMUserInstructions: WOSwitchComponent {
        WOComponentName = "SMUserInstructions";
    }
```

Mandatory Mappings

Some of these mappings are required, as the named page does not correspond to an actual page. The required mappings and the suggest name to map them to are:

LoginPage = SMLoginPage;
LogoutPage = SMLogoutPage;

Reusable Components

There are several components in GVCSMCore that are intended to be reused either as components on a page, or as a functional sub-class for a replacement page. A partial list:

Components

ChangePasswordPanel
HeaderBar
LoginPanel
SearchResultPageFlipper
SendPasswordPanel
SMUserInstructions
UnitChooser
UnitChooserWithDefault
WOSMConfirmAction

Whole Pages

SMLogoutPage
SMLoginPage

Bindings for PageScaffold and Sub-classes

The bindings (WOD file) in PageScaffold for displayed Sections are taken from PageScaffold.wod in GVCSMCore. If an installation needs customized bindings, PageScaffold.wod can be copied to GVCSMCustom and customized there. If PageScaffold.wod is found in GVCSMCustom this will be used in preference to the version in GVCSMCore. This will only be effective if the new / changed bindings can be done using only KVC and do not need custom code in PageScaffold.java. This can often be supported by using a Custom EO (see above). If custom code is needed in PageScaffold.java, then PageScaffold.java will have to be sub-classed in GVCSMCustom as CustomPageScaffold.java (for example) and CustomPageScaffold page mapped to PageScaffold. As an alternative to page mapping, the Section Style's pageScaffoldComponent attribute can be set to CustomPageScaffold instead of PageScaffold. This is also useful when the custom PageScaffold is not to be used for all styles.

Bindings for the Website Creation Message

The bindings (WOD file) for the WebsiteCreationMessage component that generates the automatic e-mail when a new Website is created are taken from WebsiteCreationMessage.wod in GVCSMCore. If an installation needs customized bindings, WebsiteCreationMessage.wod can be copied to GVCSMCustom and customized there. If WebsiteCreationMessage.wod is found in GVCSMCustom this will be used in preference to the version in GVCSMCore. This will only be effective if the new / changed bindings can be done using only KVC operating on the WebsiteCreationMessage component and do not need custom code in WebsiteCreationMessage.java. As this component has a reference to the newly created Website, additional features could be integrated into a custom sub-class of Website.

If custom code in WebsiteCreationMessage is the only solution, WebsiteCreationMessage.java will have to be sub-classed in GVCSMCustom as CustomWebsiteCreationMessage.java (for example) and CustomWebsiteCreationMessage page mapped to WebsiteCreationMessage.

Bindings for the User Creation Message

The bindings (WOD file) for the UserCreationMessage component that generates the automatic e-mail when a new external User is created are taken from UserCreationMessage.wod in GVCSMCore. If an installation needs customized bindings, UserCreationMessage.wod can be copied to GVCSMCustom and customized there. If UserCreationMessage.wod is found in GVCSMCustom this will be used in preference to the version in GVCSMCore. This will only be effective if the new / changed bindings can be done using only KVC operating on the UserCreationMessage component and do not need custom code in UserCreationMessage.java. As this component has a reference to the newly created User, additional features could be integrated into a custom sub-class of User.

If custom code in UserCreationMessage is the only solution, UserCreationMessage.java will have to be sub-classed in GVCSMCustom as CustomUserCreationMessage.java (for example) and CustomUserCreationMessage page mapped to UserCreationMessage.

Bindings for the Data Access Notification Messages

The bindings (WOD file) for the DANotificationMessage component that generates the automatic e-mail when a record is created or modified are taken partially from DANotificationMessage.wod in GVCSMCore and partially created in code. If an installation needs customized bindings, DANotificationMessage.wod can be copied to GVCSMCustom and customized there. If DANotificationMessage.wod is found in GVCSMCustom this will be used in preference to the version in GVCSMCore. This will only be effective if the new / changed bindings can be done using only KVC operating on the DANotificationMessage component and do not need custom code in DANotificationMessage.java. The DANotificationMessage component has a reference to the DataAccessParameters, which in turn have a reference to the DataAccess PageComponent. Additional features could be integrated into a custom sub-class of DataAccess.

If custom code in DANotificationMessage is the only solution, DANotificationMessage.java will have to be sub-classed in GVCSMCustom as CustomDANotificationMessage.java (for example) and CustomDANotificationMessage page mapped to DANotificationMessage.

Customizing Section Styles

The components used to render Sections can be controlled on a finer grain than simple Page Mapping allows. The PageScaffold.wod uses WOSwitchComponents which take the WOComponentName from the style. For example:

```
NavBar: WOSwitchComponent {
    currentSection = sectionToDisplay;
    WOComponentName = activeSectionStyle.navBarComponent;
}
```

The components used to display the banner, footer, navigation bar, and section contents can thus be changed on a style by style basis as long as they have the same API and only use bindings available through PageScaffold.java. This allows such things as a vertical navigation bar to be created. If extensions beyond this are needed, the style also supports using an entirely different PageScaffold component for a style. At present there is no UI support for changing these, direct database edits are required. There are also no substitute components for the banner, footer, navigation bar, or section contents.

Customizing Data Access Default Templates

The default templates used for the add record, single record, and list of records views in a Data Access Section can be customized by adding specially named HTML files to the Resources of an GVCSMCustom framework. The names of the default template files are:

DA_AddModeDefaultTemplate.html
DA_SingleModeDefaultTemplate.html
DA_ListModeDefaultTemplate.html

Customizing or Adding Mixed Media Definitions

The default templates used for the different media types in a Mixed Media section can be customized by adding specially named XML files to the MixedMedialXML directory in the Resources of the GVCSMCustom framework. The names of the default template files are:

SMMediaConfig_AVIMovie.xml
SMMediaConfig_DownloadableFile.xml
SMMediaConfig_Image.xml
SMMediaConfig_Flash.xml
SMMediaConfig_JavaApplet.xml
SMMediaConfig_QuickTimeMovie.xml
SMMediaConfig_RealAudioStream.xml
SMMediaConfig_TextOrHTML.xml
SMMediaConfig_WebPage.xml

The existing templates in the MixedMedialXML directory in the Resources of the GVCSMCore framework should be used as examples of the format and content.

New media types can be added by adding additional files to the MixedMediaXML directory in the Resources of the GVCSMCustom framework. These files can have any name, but must have an .xml extension. The applications must be restarted after adding new media types.

Customizing TinyMCE (if used as the WYSIWYG editor)

The default configuration used by TinyMCE all throughout the application is defined in one javascript. This javascript can be found in the editorSelectors directory found in /<document root>/tiny_mce. The name of the javascript is **tinymce-init-advanced.js.**

For the different configuration options for TinyMCE, you can download its documentations at http://tinymce.moxiecode.com/.

Adding Support for Groups on an LDAP Server

An EOModel needs to be added to the GVCSMCustom framework for each sub-tree containing groups on the LDAP server. Multiple sub-trees or multiple servers will require multiple models to be created. In addition, corresponding configuration items need to be added to the GVCSiteMakerBootstrap.plist file in that framework. Example models can be found in the Development version of the GVCSMCustom framework.

The Connection Dictionary

When creating the EOModel, set the adaptorName to JNDI and fill out the connection dictionary as follows:

serverUrl This is formed as ldap://<host_name>:<optional port number>/<search_base_dn>. For example, ldap://ldap.itd.umich.edu/ou=User%20Groups,ou=Groups,dc=umich,dc=edu. The <search_base_dn> is the path on the LDAP server below which the groups are found.

Note the **%20** in the URL above. The LDAP URL must be in URL-encoded format, and so the space between User and Groups has been replaced with %20.

plugInClassName The plugin class controls authentication and other behaviour. Authentication methods other than None or Simple will require the use of a custom GVC plugin.

Authentication Method	Plugin Class
None (anonymous queries)	com.webobjects.jndiadaptor.LDAPPlugIn
Simple (insecure authentication)	com.webobjects.jndiadaptor.LDAPPlugIn
Start TLS (TLS secured authentication and queries)	net.global_village.jndi.LDAPTLSPlugin
SSL (not currently implemented)	net.global_village.jndi.LDAPSSLPlugin
Kerberos (not currently implemented)	net.global_village.jndi.LDAPKerberosPlugin

timeout The default of 3600 is too long. 5 (seconds) is suggested as a more appropriate value. Searches not completing in that time will fail resulting in authorization failure. This number can be adjusted to suit the installation's needs. The GVC.SiteMaker will stall if this is too high and the LDAP server too slow.

authenticationMethod The default plugin will handle None and Simple. This setting is ignored by the GVC custom plugin classes.

username This is required when using the Simple authenticationMethod with the default plug in. It is also required for the GVC custom plugins.

password This is usually required if username is required.

initialContextFactory Leave this as the default value com.sun.jndi.ldap.LdapCtxFactory.

scope Leave this as Subtree.

The LDAP Group Entity

The model only needs to contain a single entity.

Name

The entity can be named anything as long as that name is not already in use by GVCSiteMaker. Note that the obvious name, LDAPGroup, is already in use and cannot be used in this EOModel.

Table (External) Name

The Table Name (a.k.a. externalName) for the entity should be the LDAP objectClass for the record being fetched. This is usually rfc822MailGroup.

Class

The class should be `com.gvcsitemaker.core.LDAPGroupRecord` or a sub-class of this. Using `com.gvcsitemaker.core.LDAPGroupRecord` should be fine for most situations, but you might need to create custom sub-classes in the GVCSMCustom framework to handle special situations. Consult the documentation for `com.gvcsitemaker.core.LDAPGroupRecord` to determine useful methods to override.

The LDAP Group Attributes

The entity must contain four attributes: name, relativeDistinguishedName, members, and externalMembers.

name

For the name attribute specify the LDAP group's name attribute as the column name. This will probably be just cn (common name). The External Type should be Directory String and the Internal Type should be String. The name attribute should be a locked, non-null, class property.

relativeDistinguishedName

For the relativeDistinguishedName attribute also specify relativeDistinguishedName as the column name and the External Type. The Internal Type should be String. The name attribute should be a locked, non-null, class property. The Write Format should be set to cn=<name> where cn is the Column Name used for the name attribute. As this is only used for creating new LDAP records, which GVC.SiteMaker does not do, errors in the Write Format are probably irrelevant. The relativeDistinguishedName attribute should be a primary key (and locked, non-null, non-class property).

members

For the members attribute specify the LDAP group's member attribute as the column name. The External Type should be DN (Distinguished Name) and the Internal Type should be String. The name attribute should be a locked, nullable, class property.

externalMembers

For the externalMembers attribute specify the LDAP group's attribute holding the e-mail address of other members as the column name. The External Type should be DN (Distinguished Name) and the Internal Type should be String. The name attribute should be a locked, nullable, class property.

Supporting Configuration

Once the model has been created, related configuration items need to be added to the GVCSiteMakerBootstrap.plist file. Consult the **Configuration Guide** for details. The GroupSubTree configuration item must match the <search_base_dn> used in the LDAP URL in the connection dictionary. The LDAPEntityName must match the name that you used for the entity in this model.