

*Zimlets™ – A Mechanism for Integrating
Disparate Information Systems and Content
with the Zimbra Collaboration Suite™
(ZCS)*

PRELIMINARY DRAFT

Version 0.97

Zimbra

Copyright 2004-2006 Zimbra, Inc.

All rights reserved

Table of Contents

1 Introduction 4

2 What is a Zimlet? 10

3 Zimlet Definition File..... 10

 3.1 Outline of a Zimlet Definition File (XML) 10

 3.1.1 <summary> Element..... 11

 3.1.2 <include> Element 11

 3.1.3 <includeCSS> Element..... 11

 3.1.4 <resource> Element 12

 3.1.5 <handlerObject> Element 12

 3.1.6 <contentObject> Element 12

 3.1.7 <serverExtension> Element 12

 3.1.8 <zimletPanellItem> Element..... 12

 3.1.9 Example 12

 3.2 Common Elements..... 13

 3.2.1 <actionUrl> Element 13

 3.2.2 <formEditor> Element..... 15

 3.2.3 <canvas> Element 16

 3.2.4 <contextMenu> Element..... 17

 3.3 <contentObject> Element..... 17

 3.3.1 <matchOn> Element..... 18

 3.3.2 <onClick> Element..... 18

 3.3.3 <toolTip> Element..... 19

 3.4 <serverExtension> Element 19

 3.4.1 ZimletHandler Interface 20

 3.5 <zimletPanellItem> Element 20

 3.5.1 <toolTipText> Element..... 20

 3.5.2 <doubleClicked> Element..... 21

 3.5.3 <singleClicked> Element 21

 3.5.4 <dragSource> Element..... 21

 3.5.5 ZCS Objects and Object Properties 22

 3.6 <userProperties> Element 25

4 Zimlet JavaScript Class..... 26

 4.1 Zimlet Panel Item Methods 27

 4.1.1 doDrag Method 27

 4.1.2 doDrop Method 28

 4.1.3 singleClicked Method..... 28

 4.1.4 doubleClicked Method 28

 4.2 Content Object Methods 28

 4.2.1 match Method 28

 4.2.2 clicked Method 28

 4.2.3 toolTipPoppedUp Method 28

 4.2.4 toolTipPoppedDown Method 29

 4.3 Common Methods..... 29

 4.3.1 menuItemSelected Method..... 29

 4.3.2 createPropertyEditor Method..... 29

 4.3.3 submitForm Method..... 29

 4.4 Helper Methods 30

 4.4.1 applyXslt 30

 4.4.2 checkProperties 30

 4.4.3 enableContextMenuItem Method 30

 4.4.4 getConfigProperty Method..... 30

 4.4.5 getResource 30

PRELIMINARY DRAFT

- 4.4.6 getUserPropertyInfo..... 30
- 4.4.7 getUsername 31
- 4.4.8 getUserProperty..... 31
- 4.4.9 saveUserProperties 31
- 4.4.10 sendRequest Method..... 31
- 4.4.11 setUserProperty 32
- 5 Zimlet JSP 32
- 5.1 Custom Zimbra Tag Library 32
- 6 Proxy Servlet..... 33
- 7 Messages Properties Files..... 34
- 8 Zimlet Configuration File 34
- 9 Zimlet Lifecycle and the Zimlet Management Tool 35
- 9.1 Listing Zimlets 35
- 9.2 Deploying 36
- 9.3 Configuration 36
- 9.4 Access Control 37
- 9.5 Priority 37
- 9.6 Enabling 38
- 9.7 Disabling..... 38
- 9.8 Undeploying 38
- 9.9 Getting Zimlet information 39
- 10 Example <zimlet>.xml: Maps 39
- 10.1 com_zimbra_ymaps.xml 39

1 Introduction

Zimlets™ are a mechanism for easily integrating the Zimbra Collaboration Suite (ZCS) with any Internet or intranet information system/application that provides an XML/web services bindings. We also use the Zimlet approach to “mash-up” (intermix) user interfaces within the Zimbra Collaboration Suite itself, such as by mashing up calendar and contacts within your email so that you need not switch back and forth. While the Zimlet Specification was designed for the Zimbra Collaboration Suite, the model could be easily be generalized for other Web 2.0/AJAX user interfaces.

Web 2.0 and Email

Web browsing and email/messaging are *the* two “killer applications” of the Internet. Many (or even most?) power users devote more time to their messaging/calendaring application than any other networked application, and yet that messaging/calendaring application itself has typically failed to keep pace with the technology advances of the Internet.

The Web is a “pull” experience in that the user initiates synchronous requests for content or services, while email and messaging (instant messaging, voicemail, etc.) are “push” experiences in that content from others (often too much content) is asynchronously delivered (or “pushed”) to the user. For the user, the separation between the browsing (pull) and messaging (push) paradigms can be frustrating—how much user time is wasted cutting and pasting content from email into the browser in order to take action based on message content? If we can successfully blend more web functionality into messaging applications, we can both improve user productivity for email and reduce the gap in user experience between push and pull.

Web 2.0 technologies like Asynchronous JavaScript and XML (AJAX, which is the topic of another Zimbra whitepaper¹) and XML/web services, and next-generation collaboration solutions like Zimbra that leverage AJAX and XML, enable this blending of “push” and “pull” within a single unified model. Web 2.0 (buzzword “*du jour*”) really encapsulates three things:

- Richly-interactive user interface in the browser – With technologies like AJAX, web user interfaces can be every bit as rich as that of traditional client/server applications, but at the same time still have the look, feel, behavior, and zero client administration of the World-Wide Web;
- XML-based web services – Service-Oriented Architecture (SOA) based on XML/web service end-points allows Internet data and services (maps, stock quotes, weather, flight status, etc.) and intranet data and services (enterprise applications such as CRM, ERP, etc.) to be securely accessed from other applications and systems; and
- Mash-ups – Mash-ups are the aggregation and customization of multiple web user interfaces and web services to eliminate context switching between existing systems, deliver wholly new experiences, or almost the range of everything in between.

Zimlets, as we shall see shortly, are an extensible mechanism for marrying Web 2.0 technologies to enterprise messaging (e.g., email, IM, voice) and collaboration. With Zimlets, arbitrary message content can be made “live” by linking it with web content and services on intranets or the Internet. No more cutting and pasting from email to browser. “Mousing” over actionable content gives the user a real-time preview (subject to security constraints) that can be factored in decision making:

- Mouse-over a date or time, and see what’s in your calendar;

¹ *The Zimbra AJAX Toolkit -- A Toolkit for Developing Rich, Browser-based Applications* (http://www.zimbra.com/downloads/zimbra_ajax_tk_whitepaper.pdf)

PRELIMINARY DRAFT

- Mouse-over a contact, and see the phone number in your address book;
- Mouse-over an physical address, and see a map or even driving directions and estimated arrival time (and in the future, current traffic conditions and nearby WiFi coffee shops);
- Mouse-over a flight number, and see whether or not it's on time;
- Mouse-over a item number/catalog entry, and see what inventory and order status looks like;
- Mouse-over a customer email address or bug/case tracking number, and see its status and priority;
- Mouse-over an equity to get a quote (or the aggregate position across a trading desk);
- Mouse-over an Internet order, and see its billing and shipping status; and so on.

With Zimlets, email (as well as IM, calendaring, etc.) content can also be fully and securely actionable:

- Right click on a phone number to make a call with your soft-phone (such as via Skype or a Cisco VoIP phone);
- Drag an email to an SMS icon to edit/forward to your co-worker's phone;
- Drag an email, contact, or appointment to an IM session with a buddy;
- Right click on a bug/customer case to escalate its priority;
- Drag a song or book title to iTunes, Amazon, etc. to initiate a purchase;
- Drag a message, conversation (collection of messages), contact, or meeting summary to your CRM system icon (such as Salesforce.com or SugarCRM) in order to archive and share that data across your sales team;
- Cut and paste a word to Wikipedia to review the definition;
- Drag a contact to Yahoo! or Google maps to see where they are located;
- Right click on a name, address, or phone number to update your address book;
- Right click on a date to schedule a meeting;
- Drag an email to your calendar to schedule a meeting for participants on the thread (and share the annotated email conversation as background);
- Right click on a airline reservation to print your boarding pass;
- Right click on a part number to place an order for more inventory;
- Right click on a purchase order, provisioning request, or other internal workflow request to approve or reject it; and so on again.

What Makes Zimlets Different

For years, email applications have been highlighting and linking the text of obvious web content—such as URLs and email addresses. When a user clicks on the highlighted text, the email application takes the appropriate action:

- Launching a browser window to open the web page associated with the URL, or
- Opening the email editor so that the user can fill in the body of a message destined for a particular email address.

The beauty of this approach is that all the “smarts” are on the receiving end—the author of the email is not obliged to “tag” something as a URL or email address. Rather the receiving email application simply recognizes these magic strings and then links them for ease of user action.

Unfortunately, this mechanism has generally not been extensible: the user gets precisely what linking the email system supports “out of the box”, and extensibility is precisely where the real productivity pay-off lies. Imagine if:

- Email from customers could be auto-linked to the customer support and CRM systems;
- Email from suppliers and redistributors could be auto-linked to inventory management,

PRELIMINARY DRAFT

- manufacturing control, and accounting systems;
 - Email regarding a particular derivative could be auto-linked to trading and risk management systems; and
 - Email regarding a new employee could be auto-linked to the appropriate HR systems.
- The same holds true for insurance claims, provisioning requests, retail sales reports, and so on.

There have been a couple of prior approaches focused on *extensible* linking and actionable content. Most placed the burden of linking the content on the author of the email. This is problematic for emails that originate with customers, business partners, or even employees on their home computers, all of which often rely on email software that is other than the end user's internal standard. Far better is to have the smarts on the receiving end so that actionable messaging works with for all in-bound email, IMs, appointments, etc. whatever the source!

Other prior approaches to extensible linking and actionable content, such as Microsoft's Smart Tags, were developed for Office desktop applications (Excel, Word, Outlook, etc. as well as Internet Explorer/IE). Approaches that rely on such client-side installation and execution of the linking and action defining code are problematic in that they do not offer:

- **Multi-browser, multi-operating system** – The Zimlet approach will work in any standard Web browser (IE, Mozilla Firefox, Apple Safari, and so on) and on any client operating system (Windows, Linux, MacOS, and so on) without requiring the a priori *installation* of any client-side software. (Zimlet AJAX code extensions can, however, be dynamically and securely downloaded with the Zimbra Collaboration Client for interpretation by the Web browser.)
- **Multi-client** – Zimlets are ultimately designed to work across the range of client devices that support the Web standard platform such as mobile phones, PDAs, tablet PCs, set-top boxes, and so on. Any client device that supports the Internet-standard Web model of AJAX works is compatible with the Zimlet architecture.
- **Object recognition across mailboxes** – With the server-side execution of Zimlets, actionable objects are recognized on the server which permits them to be indexed for *search/discovery across mailboxes* such as is required for Sarbanes-Oxley or Human Resources compliance, fiduciary isolation of business units (such as automatically recognizing equity-related communications), and so on.
- **Reduced administrative overhead** – Installing and maintaining client-side DLLs is expensive and error-prone. Zimlets that are deployed within a Web 2.0 framework like the Zimbra Collaboration Suite can be automatically updated with each browser session and therefore have zero administrative overhead on the client side.
- **Security** – Client-side DLLs are also a security risk, particularly when they are permitted to access client-side data or arbitrary services on the Internet. Zimlets instead run under the protected execution model of the browser and hence are precluded from touching client data or from communicating with websites other than the web front-end for the Zimbra Collaboration Suite itself (which securely proxies requests to other web services and user-interfaces). Moreover, the server-side code for Zimlets is deployed as Java managed code which provides more protected execution from errant Zimlets inadvertently or maliciously interfering with operations.

Zimlets

Zimlets are the first actionable content model designed to leverage Web 2.0 technologies—AJAX, XML/web services, and mash-ups. Most Zimlets can be specified declaratively in a simple XML definition file that specifies the UI, behavior, and URL endpoints associated with the Zimlet. While Zimlets allow the definer to “drop” into JavaScript (client-side) or Java (server-side) code in order to provide rich, highly customized behavior, the vast majority of Zimlets are “weekend projects” that are fully characterized via that declarative template.

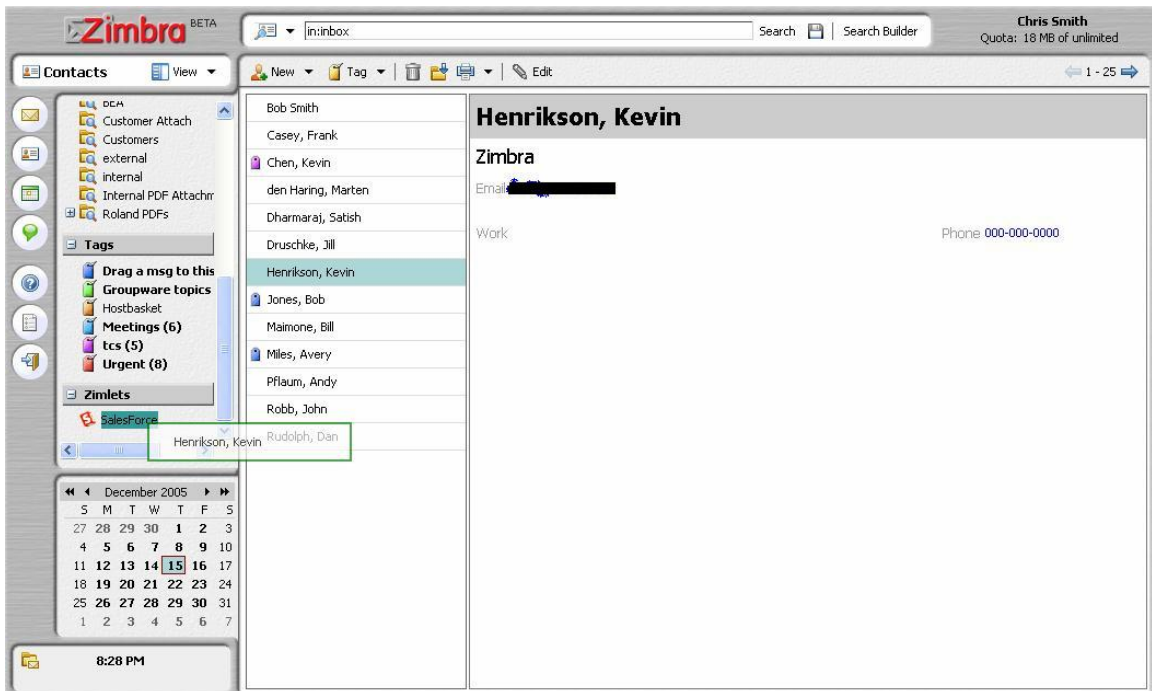
PRELIMINARY DRAFT

At the same time, Zimlets are also designed to address enterprise deployment constraints:

- Under Zimlets, the deployment is done on the server-side, where extensions/customizations can be centrally installed and managed by the system administrator.
- Security, too, is managed on the server-side under the AJAX –based Zimlet model; a client’s requests are routed via a secure, managed server that can proxy the client’s security privileges as appropriate.
- Zimlets run under a protected execution model (much like JSPs and Servlets) in a manner that prevents them from inadvertently or maliciously interfering with other processing on the system.

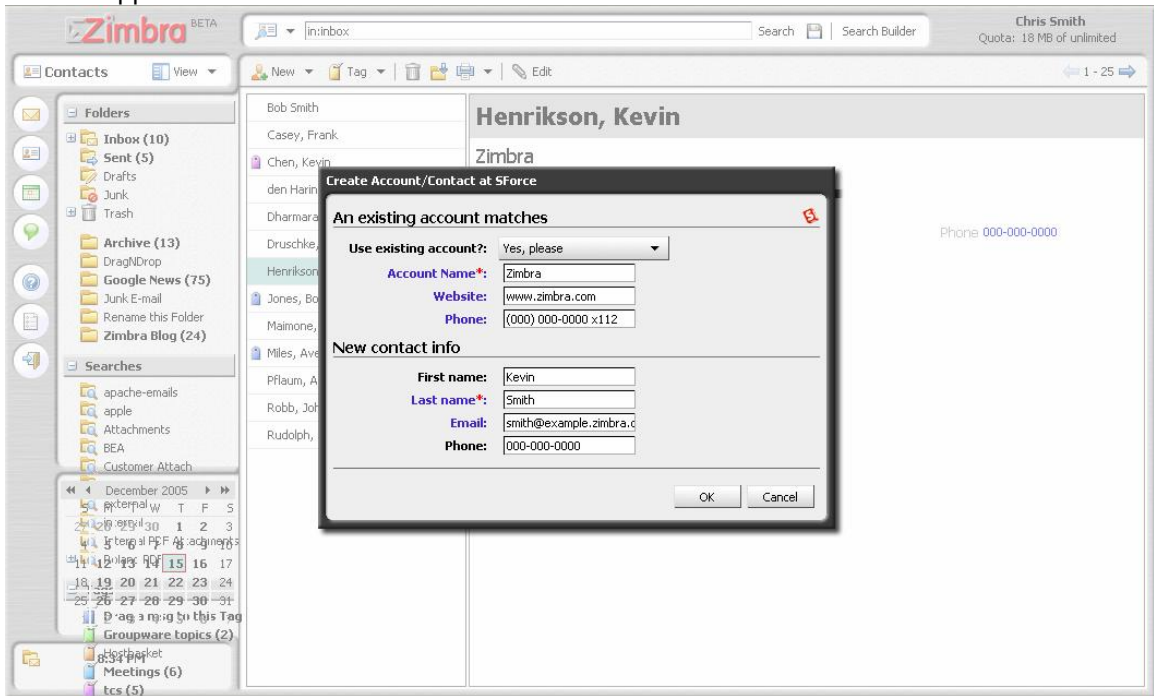
Zimlets are realized in the client in two ways:

- As panel item elements in the application overview panel. The user may interact with Zimlet panel items by dragging content such as mail messages, contacts and calendar appointment onto them, double clicking them, and invoking actions from a context menu, if one is provided by the Zimlet author. The two images below show the result of a user dragging and dropping a contact onto the “Salesforce” Zimlet panel item.

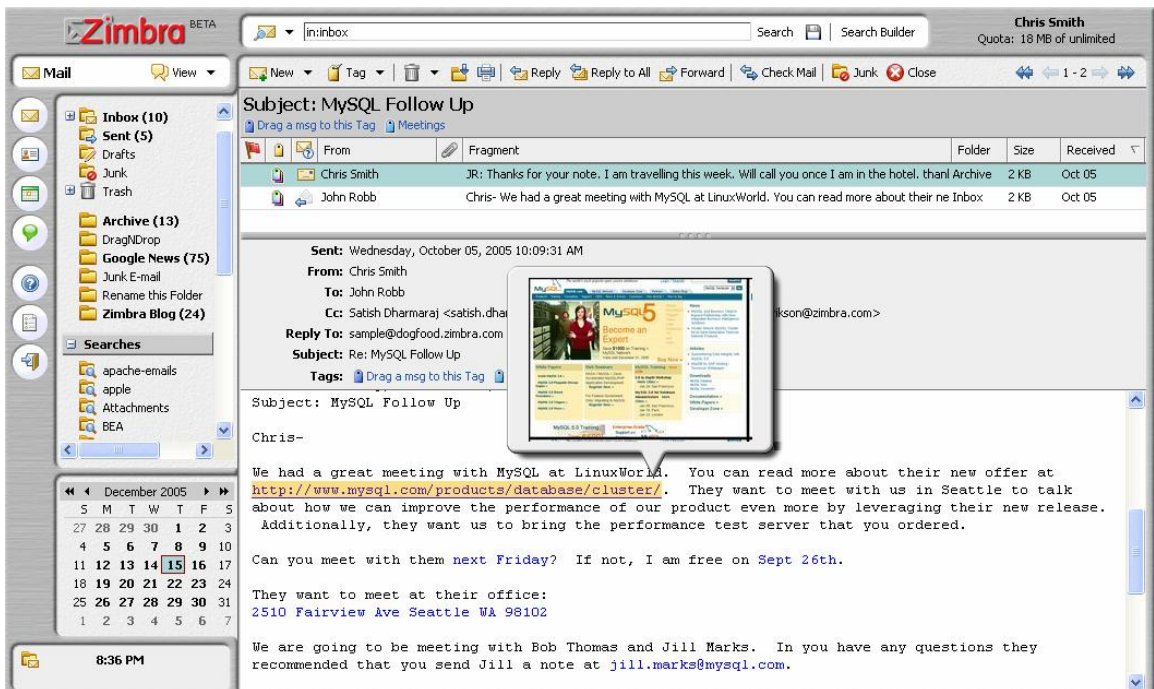


PRELIMINARY DRAFT

The image below shows what happens after the dragged item (a contact in this instance) is dropped on to the Zimlet:



- As Content objects, such as phone numbers, purchase order numbers, and URLs in content such as email message bodies, contacts, and calendar appointments. The image below shows the result of hovering over a URL Zimlet content object in the body of a mail message.



PRELIMINARY DRAFT

The Zimlet architecture was designed with the goal of enabling a broad range of Zimlets, from those requiring little or no custom code (i.e. entirely declarative) to those which use JavaScript to tightly integrate rich UI behavior within the Zimbra Collaboration Suite AJAX client.

The Zimlet architecture consists of a client and server framework, as well as a set of web services and JavaScript APIs. Zimlets are exposed to the end user via the ZCS AJAX client. The Zimlet server infrastructure provides the mechanism for lifecycle management such as deployment, configuration, access control, and “undeployment” of Zimlets. In addition, the framework provides the ability to integrate Zimlets with the ZCS search engine. The Zimlet client framework helps realize Zimlets in the ZCS AJAX client. The framework provides support for drag and drop, context menus on both panel items and content objects, and for embedding of Zimlet content into the application. The details of these will be presented within this whitepaper.

ZCS comes preconfigured with several Zimlets that provide illustrations of what is possible. These include: Integration with Yahoo/Google Maps, Alexa, Amazon, Bugzilla, SMS, Wikipedia, ZCS calendar, ZCS address book, and Skype.

The Zimbra Community website will serve as a clearing house for innovative Zimlets that various independent developers and enterprises devise. We hope to bundle the most compelling, and generally useful 3rd-party Zimlets into future releases of the Zimbra Collaboration Suite.

2 What is a Zimlet?

A Zimlet is a “zipped” bundle of content that is deployed to the ZCS server by the Zimlet management tool (described in section 9). The files comprising the Zimlet bundle enable the integration of the ZCS platform with disparate information systems and content. These files are enumerated below. Note that the “<zimlet>” label in descriptions below would be replaced with the name of the actual Zimlet. The naming convention is the reversed domain name of the author’s company where the “.”s in the domain name are replaced by “_”s. For example, if the author’s company is acme.com and the Zimlet name is Maps, then the fully qualified name would be com_acme_Maps

- **<zimlet>.xml** – The Zimlet definition file. This is the most important file in the Zimlet bundle. It specifies the behavior of the Zimlet. In fact, it is possible to fully specify this behavior in the Zimlet definition file without the need for writing any client side JavaScript code. This file is required for all Zimlet implementations.
- **<zimlet>.js** [optional] – While it is possible to completely define a Zimlet declaratively within the Zimlet description file, in many instances it may be desired to provide JavaScript implementation for some or all of the Zimlet functions. The Zimlet JavaScript class provides the base class whose methods may be overridden to provide custom implementations for various Zimlet functions (see section 4 for a full description of the Zimlet base class). This file is only required for those Zimlets requiring the extension of the AJAX client with custom JavaScript code. <zimlet> can actually be any name though we recommend using the convention above.
- **<zimlet>.jsp** [optional] – This file contains optional JSP code to implement connectivity, data retrieval, or additional server-side data functionality. Only Zimlets requiring custom server side implementations need provide a JSP implementation. The Zimlet infrastructure comes with a generic proxy servlet rooted at “/service/proxy” for making arbitrary calls to external systems or services. See section 6 for more information.
- **<zimlet>.properties** [optional] – This is the fallback Zimlet message property file. By using Java property files, Zimlets are easily Localizable. See section 7 for details.
- **config_template.xml** [optional] – This is a template file for the Zimlet’s configuration properties for the Zimlet. See section 8 for more information.

In addition, the Zimlet bundle may also include:

- Additional supporting JavaScript files
- A Java class that implements the `com.zimbra.cs.zimlet.ZimletHandler` interface to help match Zimlet content objects for server-side indexing (see section 3.4 for more information)
- Additional properties files for the locales for which the Zimlet has been localized.
- Additional supporting files such as CSS, HTML, image files, etc.

3 Zimlet Definition File

This section describes the Zimlet definition file in detail.

3.1 Outline of a Zimlet Definition File (XML)

The <zimlet> element is the enclosing element in the definition file. The <zimlet> element has the following attributes:

- **name** - The name attribute is the Zimlet's name and is a required attribute. This must be the same name as is used to prefix all the well defined files in the Zimlet bundle. The Zimlet name is required to be unique within a deployment. The naming convention for Zimlets follows that of Java. That is, the name is prefixed with the domain of the author's company; however, the "."s in the domain are replaced by "_" characters. From example: name="com_acme_Maps".
- **version** - This required attribute specifies the Zimlet's version. For example, version="1.0".
- **description** – This attribute provides a short (approximately one line) description used in mouse-overs and dialogs.

The <zimlet> element may also contain a number of optional child elements that define the Zimlet. The following sub sections describe each of these child elements.

3.1.1 <summary> Element

The summary element contains text describing the Zimlet in more detail. It can also contain limitations and a list of current functionality:

```
<summary>
  This Zimlet will provide map information for street addresses
  Note: Currently maps are not scrollable or clickable.
</summary>
```

The summary element is an optional element.

3.1.2 <include> Element

In certain cases a Zimlet implementation may require additional JavaScript code to be loaded by the client. For such cases, the <include> element is provided. The content of this element provides a reference to the additional script files that are to be loaded. These files may be internal, that is specified by a relative URL (e.g. foo.js) or external, that is specified by an absolute URL (e.g. http://api.google.com/maps.js). If multiple <include> elements are provided, loading order is maintained (top to bottom).

Below are examples of the <include> element:

```
<include>http://api.google.com/maps.js</include>
<include>gmaps.js</include>
```

3.1.3 <includeCSS> Element

A Zimlet implementation may require CSS styles to be loaded by the client. The <includeCSS> element is provided for such cases. The content of this element provides a reference to the additional CSS file that is to be loaded. This file may be internal, that is specified by a relative URL (e.g. foo.css) or external, that is specified by an absolute URL (e.g. http://my.server.com/styles.css). Any CSS file included with a <includeCSS> element is automatically loaded. It is required that any CSS class in these files be prefixed with or include the zimlet name to avoid collisions with other CSS classes or Zimlets.

Below are examples of the <includeCSS> element:

```
<includeCSS>http://my.server.com/styles.css</includeCSS>
<includeCSS>styles.css</includeCSS>
```

3.1.4 <resource> Element

A Zimlet implementation may require additional resources: images, binaries, etc. For such cases, the <resource> element is provided. The content of this element provides a reference to the additional files that are to be loaded. These files may be internal (specified by a relative URL, e.g. image.gif) or external (specified by an absolute URL, e.g. http://api.google.com/maps.jgif). Resources are not automatically loaded. They can be accessed with the fully qualified URL via the Zimlet JavaScript class' `getResource()` method (see section 4.4.8)

3.1.5 <handlerObject> Element

The <handlerObject> element designates the constructor to call in the <zimlet>.js at init time. The <handlerObject> element may be omitted if the zimlet does not implement custom JavaScript code.

3.1.6 <contentObject> Element

A content object is an object that is recognized by a Zimlet in a body of content, such as an email message body, the notes of an appointment, or fields in a contact. Examples of content objects are: Phone numbers, URLs, purchase order numbers, package tracking numbers, etc. The <contentObject> element defines the content object that a Zimlet recognizes and how the user interacts with the object. If a Zimlet does not expose a content object then this element may be omitted. The <contentObject> element is fully described in section 3.3

3.1.7 <serverExtension> Element

The <serverExtension> element describes how a content object is indexed on the server. Content, such as email message bodies, calendar appointment notes, and contact fields, that contains one or more Zimlet content objects may be searched for via the query language's "has" clause. For example, if a Zimlet were to define phone numbers as a content object, then to search for content containing phone numbers, a <serverExtension> element would be specified that defines phone numbers, and would enable searching for content containing phone numbers via the "has:phone" clause. The <serverExtension> element is fully described in section 3.4

3.1.8 <zimletPanelItem> Element

The Zimlet panel appears within the ZCS AJAX client's overview panel. The overview panel is to the right of the application buttons and also contains panels for a user's folders, tags, and saved searches. By defining a <zimletPanelItem> element, a Zimlet may add an item to the Zimlet panel within the overview panel. The user may interact with a Zimlet panel item by clicking, double clicking, right-clicking for a context menu, as well as by dragging and dropping various objects, such as contacts, conversations, and mail messages onto the panel item. As part of the panel item definition, the Zimlet author specifies the type of objects that the Zimlet supports as well the actions to be performed on those objects. If a Zimlet author does not wish to expose a panel item for their Zimlet, then this element may be omitted. The <zimletPanelItem> element is fully described in section 3.1.8

3.1.9 Example

Below is an example outline of a Zimlet definition file:

```
<zimlet name="com_zimbra_sforce" version="1.0"  
      description="Salesforce INTEGRATION">
```

PRELIMINARY DRAFT

```
<include>sforce.js</include>
<includeCSS>sforce.css</includeCSS>

<handlerObject>Com_Zimbra_SFforce</handlerObject>

<zimletPanelItem label="Salesforce" icon="SFORCE-panelIcon">

  <toolTipText>
    Use drag'n'drop to interact with your Salesforce account
  </toolTipText>

  <dragSource type="ZmContact" />
  <dragSource type="ZmMailMsg" />
  <dragSource type="ZmConv" />

  <contextMenu>
    <menuItem label="Login" id="LOGIN" />
    <menuItem label="Preferences..." id="PREFERENCES" />
    <menuItem />
    <menuItem label="Visit Salesforce" id="WWW.SFORCE.COM"
      icon="SFORCE-panelIcon">
      <canvas type="dialog" title="Salesforce Website"
        width="800" height="600" />
      <actionUrl target="http://www.sforce.com">
        <param name="referrer">www.zimbra.com</param>
      </actionUrl>
    </menuItem>
  </contextMenu>

</zimletPanelItem>

<userProperties>
  <property type="string" name="user" minLength="4" maxLength="32"
    label="User ID" />
  <property type="password" name="passwd" minLength="4"
    maxLength="32" label="Password" />
</userProperties>

</zimlet>
```

3.2 Common Elements

There are several elements that may appear in various contexts in both the `<contentObject>` and `<zimletPanelItem>` elements. This section describes these elements.

3.2.1 `<actionUrl>` Element

The `<actionUrl>` element describes a URL that is called based on a given action. Examples of actions are clicking on a content object or dropping a contact onto a Zimlet panel item. The `<actionUrl>` element has several attributes:

- `method` - The value of this attribute may be either "get" or "post". If omitted, it defaults to the HTTP "get" method.
- `target` - This attribute specifies the target URL. This may be either an absolute URL or a relative URL.
- `xslt` - XSLT style sheet filename. This file must be part of the Zimlet bundle. If present

PRELIMINARY DRAFT

the XSLT style sheet is applied to the result of the target. The output of applying the style sheet must be an HTML element such as a div.

- paramStart – This attribute specifies the character to use when appending parameters to a GET URL. If omitted, the default is “?”.
- paramJoin – This attribute specifies the character to use when appending additional parameters. If omitted, the default is “&”.

In addition to the above attributes, the <actionUrl> element can have zero or more <param> element children.

The <param> element specifies parameters that are to be passed to the target URL. The <param> element has a single required “name” attribute that specifies the parameter name. The content of the element is the value of the parameter. The possible content of the <param> element varies depending on the context in which the <actionUrl> element is being defined as follows:

- <contentObject>
 - The value of the content object represented by the following:
\${src.objectContent}
 - Arbitrary string content with or without \${src.foo} type variables.
- <zimletPanelltem>
 - A ZCS object property. The valid set of object properties for the action depends on the object being dropped onto the panel Item. Object properties are prefixed with ‘\${obj.’ and suffixed with ‘}’ for example, \${obj.cid}. ZCS objects and their properties are described in section 3.5.5).
 - Arbitrary string content with or without \${src.foo} type variables

Additionally if the <actionUrl> element appears alongside a <formEditor> element, then the values of the form fields (see 3.2.2) are available to the <actionUrl>. Form fields are prefixed with ‘\${form.’ and suffixed with ‘}’ for example, \${form.quantity}

The Zimlet framework will package and send the parameters according to the value specified by the enclosing <actionUrl> element’s “method” attribute.

The following is an example of a <actionUrl> element that may be defined for an action on a content object:

```
<actionUrl method="get" target=http://www.mapquest.com>
  <param name="address">${src.objectContent}</param>
</actionUrl>
```

Assuming the value for \${src.objectContent} is “111 Main St Anytown 94044”, then the Zimlet framework would send the following data to the server:

```
http://www.mapquest.com?address=111%20Main%20St%20Anytown%2094044
```

Here is another example with static param value:

```
<actionUrl method="get" target="http://www.amazon.com">
  <param name="referrer">www.zimbra.com</param>
</actionUrl>
```

When inside a <contentObject> element, <param>-eters also support \${src.\$1}, \${src.\$2}, etc. These will get replaced by contents of the N-th parentheses group in a matched string. Let’s take

the “Bugz” Zimlet as an example. It matches text such as “bug 1234”, therefore `${src.objectContent}` will be “bug 1234”. However, when the user clicks on the content object, it should open the bug URL from Bugzilla, so it needs `_only_` the bug ID. Therefore, the matching RegExp can be stated as:

```
<regex attr="ig">bug\s+([9-0]+)</regex>
```

and the `<actionUrl>` can be stated as:

```
<actionUrl target="http://bugzilla.zimbra.com/show_bug.cgi">  
  <param name="id">${src.$1}</param>  
</actionUrl>
```

3.2.2 `<formEditor>` Element

A form editor permits the definition of a form into which the user enters information that may be submitted to a remote server. The `<formEditor>` element specifies the following required attribute:

- `name` – This is the name of the form. This name must be unique within the Zimlet definition file.

The `<formEditor>` element may contain one or more `<field>` elements. There are several types of field elements, and they all share the following set of common attributes:

- `type` - The type of the field such as “string”. The set of valid fields is described below.
- `name` - The field name.
- `label` - The label to be displayed for the field in the form editor.
- `value` - The default value for the field (optional).

If the `<formEditor>` element appears alongside an `<actionUrl>` element, then the fields of the form may be accessed in the `actionUrl` as described in section 3.2.1. If there is no `<actionUrl>` element specified, then the Zimlet JavaScript class `submitForm()` method is called (see section 4.3.3)

The available set of form field types are described in the following subsections.

number type

The number type represents a number. When a field’s type attribute is “number”, then the following optional attributes are available:

- `minVal` - The minimum value permitted for the field
- `maxVal` - The maximum value permitted for the field

string type

The string type represents a string. When a field’s type attribute is “string”, then the following optional attributes are available:

- `minLength` - The minimum length of the string
- `maxLength` - The maximum length of the string
- `visualType` – May be either “inputField” or “textArea” corresponding to an input field or a multi-line text area respectively. If omitted it defaults to “inputField”
- `numLines` – Applicable only if the `visualType` is “textArea”. Specifies the number of lines that should be displayed in the text area.

password type

The password type represents a password. A password is typically obfuscated as the user types it in. When a field's type attribute is "password", then the following optional attributes are available:

- minLength - The minimum length of the string
- maxLength - The maximum length of the string

date type

The date type represents a date. Dates are represented internally as milliseconds since the beginning of the Unix epoch.

boolean type

The Boolean type represents a "true" or "false" value. This is presented to the user as a check button.

enum type

The enum type represents a set of choices. This may be presented to the user as a dropdown box with a list of the valid choices or a set of radio buttons. A field of this type has one or more <item> child elements. The <item> element defines three attributes:

- label - The user interface label for the element. This is a required attribute.
- value - The value for the item. This is a required attribute.
- visualType - This may be either "select" or "radio" corresponding to a drop down select element or a set of radio buttons respectively. If omitted, then this value defaults to "select"

<formEditor> Example

```
<formEditor name="smsForm">
  <field type="string" name="phoneNo" label="{msg.phoneNo}"
    maxLength="14"/>
  <field type="string" name="message" label="{msg.message}"
    visualType="textArea" maxLength="120" numLine="10"/>
</formEditor>
```

3.2.3 <canvas> Element

Certain actions may result in content being presented to the user. In certain cases this content is static, e.g. driving directions; in other cases it may be interactive, e.g. an html form. The <canvas> element describes the visual element into which such content is to be rendered.

- type - This required attribute defines the type for the canvas. This may be one of:
 - none - There is no visual element for this action.
 - dialog - The visual will be a dialog.
 - tooltip - The visual will be a tool tip.
 - window - A new browser window will be created.
- width - The width in pixels of the visual. This is an optional parameter and is ignored if the visual type is "none". If this attribute is omitted, then the Zimlet framework will automatically size the content.
- height - The height in pixels of the visual. This is an optional parameter and is ignored if the visual type is "none". If this attribute is omitted, then the Zimlet framework will automatically size the content

If this element is omitted from an action, it is the equivalent of specifying a <canvas> element with a type attribute value of "none"

3.2.4 <contextMenu> Element

A context menu is usually popped up when the user right clicks the mouse over a content object or a panel item (should either be defined). The <contextMenu> element specifies the context menu items and their corresponding actions. The <contextMenu> element is optional and omitting it will disable the context menu for the Zimlet's content object. If present, the <contextMenu> element must contain one or more <menuItem> elements.

<menuItem> Element

A <menuItem> element describes an entry in the context menu. A <menuItem> element defines the following attributes:

- id - A unique identifier for the menu item. This is a required attribute
- label - This is the text that will be displayed for the menu item in the context menu
- icon - The url of the icon to be displayed for the menu item (optional)
- disabledIcon - The url of the icon to be displayed when this menu item is disabled (optional)

It may contain the following child elements:

- <canvas> - determines the visual for the menu item. The <canvas> element is described in section 3.2.3
- <actionUrl> - Specifies the URL to load. The <actionUrl> element is described in section 3.2.1. If this element is not present, then the Zimlet framework will call the Zimlet class' menuItemSelected method (described in section 4.3.1). Note that the only way to enable or disable a menu item is via the JavaScript methods.

<contextMenu> Example

```
<contextMenu>
  <menuItem label="{msg.mapquest}" icon="mapquest.gif" id="MAPQUEST">
    <canvas type="window"/>
    <actionUrl target="http://www.mapquest.com">
      <param name="">{src.objectContent}</param>
    </actionUrl>
  </menuItem>

  <menuItem label="{msg.yahooMaps}" icon="yahoo.gif" id="YAHOOMAPS">
    <canvas type="dialog" width="200" height="200"/>
  </menuItem>
</contextMenu>
```

3.3 <contentObject> Element

A content object is an object that is recognized by a Zimlet in a body of content, such as email message body, the notes of an appointment, or fields in a contact. Examples of content objects are: Phone numbers, URLs, purchase order numbers, package tracking numbers, etc. The <contentObject> element defines the content object that a Zimlet recognizes and how the user interacts with the object. If a Zimlet does expose a content object, then this element may be

omitted.

The <contentObject> element may contain a number of child elements which will be described in the following sections.

3.3.1 <matchOn> Element

Much as the <serverExtension> element specifies the rules that must be applied to match a content object for server side indexing, the <matchOn> element specifies the rules for the matching content objects on the client. There are two mechanisms that may be used to define these rules:

1. The first is with the <regex> element. The content of this element specifies a regular expression that defines the Zimlet's content object. The format of this regular expression must follow the **JavaScript** regular expression grammar. The attrs attribute can be used to set any RegEx attributes.
2. If the content object matching rules cannot be expressed by a regular expression, then the Zimlet class' match method (described section 4.2.1) may be overridden to provide the matching rules

<matchOn> Example

```
<matchOn>
  <regex attrs="ig">1[zZ]\\s?\\w{3}\\s?\\w{3}\\s?\\</regex>
</matchOn>
```

3.3.2 <onClick> Element

The <onClick> element defines the behaviour for the click action. This usually means the user clicking on a content object with the left mouse button. <onClick> may have the following elements:

- <canvas> - determines the visual for the action. The <canvas> element is described in section 3.2.3
- <actionUrl> - Specifies the URL to be loaded. The <actionUrl> element is described in section 3.2.1. If this element is omitted, then the Zimlet framework's will call the Zimlet class' clicked method (described in section 4.2.2)

<onClick> Example

In the example below, when the user clicks on the Zimlet content object, a new browser window of 300x300 pixel dimension is opened. The browser is loaded with the content of the URL specified in the <actionUrl> element

```
<onClick>
  <canvas type="window" width="300" height="300"/>
  <actionUrl method="get" target="http://maps.google.com">
    <param name="">${src.objectContent}</param>
  </actionUrl>
</onClick>
```

3.3.3 <toolTip> Element

When a user hovers their mouse over an object, a tool tip may be popped up. The <toolTip> element controls the behaviour of this tool tip. The <toolTip> element has the following attributes:

- sticky - This Boolean attribute may be set to “true” or “false”. If omitted, it will default to “false”. If set to “true”, then the tool tip becomes a sticky tool tip. Such a tool tip behaves as a standard tool tip (i.e. will be dismissed on the appropriate amount of mouse motion) unless the user clicks within its content, at which time it will remain “popped up” requiring the user to explicitly dismiss it. Sticky tool tips are a hybrid between dialogs and standard tool tips. They are useful when the user is able to interact with the content of the tool tip - for example to zoom into the map of an address. If the sticky attribute is set to false, then the tool tip behaves as a traditional tool tip and will be dismissed when the user moves their mouse.
- width - The width in pixels of the tool tip content. If this attribute is omitted, then the Zimlet framework will automatically size the tool tip.
- height - The height in pixels of the tool tip content. If this attribute is omitted, then the Zimlet framework will automatically size the content

The <toolTip> element may also contain the following child element:

- <actionUrl> - Specifies the URL to be loaded. The <actionUrl> element is described in section 3.2.1. If this element is omitted, then the Zimlet framework will call the Zimlet class’ toolTipPoppedUp method (described in section 4.2.3). If the tool tip is a sticky tool tip, then when it is dismissed the Zimlet class’ toolTipPoppedDown method (described in section 4.2.4) will be called.

<toolTip> Example

```
<toolTip sticky="true" width="100" height="200">
  <actionUrl method="get" target="http://maps.google.com">
    <param name="">${src.objectContent}</param>
  </actionUrl>
</toolTip>
```

3.4 <serverExtension> Element

To have the ZCS server provide an index for a contentObject defined by a Zimlet, a <serverExtension> element must be provided. This element has a number of important attributes which determine how indexing takes place on the server:

- hasKeyword - This attribute specifies the keyword that will be used with the query language’s “has” clause e.g. “has:address”, or “has:id”.
- regex - This attribute specifies a regular expression that defines the Zimlet’s content object. The format of this regular expression must follow the **Java Development Kit (JDK)** regular expression grammar. This attribute is mutually exclusive with the extensionClass attribute below
- extensionClass - If the content object matching rules cannot be expressed by a regular expression, then the extensionClass attribute may be specified. The content of this attribute specifies a Java class that extends the `com.zimbra.cs.zimlet.ZimletHandler` interface. This class file must be part of the Zimlet bundle

<serverExtension> Example

```
<serverExtension hasKeyword="address"  
    extensionClass="com.acme.Matcher"/>
```

3.4.1 ZimletHandler Interface

When a regular expression is insufficient for identifying a Zimlet's content object, the Zimlet developer may provide a Java class that implements the `com.zimbra.cs.zimlet.ZimletHandler` interface with the `match()` method as shown below:

```
public String[] match(String text)
```

Parameter:

- *text* - The text to be parsed for content objects

Return Value:

- array of String containing content objects.

3.5 <zimletPanelItem> Element

The Zimlet panel appears within the ZCS AJAX client's overview panel. The overview panel is to the right of the application buttons and also contains panels for a user's folders, tags, and saved searches. By defining a `<zimletPanelItem>` a Zimlet may add an item to the Zimlet panel within the overview panel. The user may interact with a Zimlet panel item by doubling clicking it, as well as by dragging and dropping various objects, such as contacts, conversations, and mail messages onto it. As part of the panel item definition, the Zimlet author specifies the types of objects that the Zimlet supports as well the actions to be performed on those objects. If a Zimlet author does not wish to expose a panel item for their Zimlet, then this element may be omitted.

The `<zimletPanelItem>` defines the following attributes:

- *label* - The label will be displayed as the text label for the Zimlet panel item
- *icon* - The url of the icon to be displayed as the Zimlet panel item's icon (optional)

The `<zimletPanelItem>` element may contain a number of child elements which will be described in the following sections.

3.5.1 <toolTipText> Element

This optional element defines the tool tip text to be displayed when a user hovers their mouse over the Zimlet panel item

<toolTipText> example

```
<toolTipText>${msg.panelTooltipText}</toolTipText>
```

<toolTipText>Drag and drop a contact</toolTipText>

3.5.2 <doubleClicked> Element

The <doubleClicked> element defines the behavior for the double click action. This usually means the user double clicking the left mouse button on the Zimlet panel item. The <doubleClicked> element may contain the following elements:

- <canvas> - Determines the visual for the action. The <canvas> element is described in section 3.2.3
- <actionUrl> - Specifies the URL to loaded. The <actionUrl> element is described in section 3.2.1. If this element is omitted, then the Zimlet framework will call the Zimlet class' doubleClicked method (described in section 4.1.4)

<doubleClicked> Example

```
<doubleClicked>
  <canvas type="window" width="300" height="300"/>
  <actionUrl method="get" target=http://maps.google.com/>
</doubleClicked>
```

3.5.3 <singleClicked> Element

The <singleClicked> element defines the behaviour for the single click action. This usually means the user single clicks the left mouse button on the Zimlet panel item. <singleClicked> may have the following elements:

- <canvas> - Determines the visual for the action. The <canvas> element is described in section 3.2.3
- <actionUrl> - Specifies the URL to be loaded. The <actionUrl> element is described in section 3.2.1. If this element is omitted, then the Zimlet framework will call the Zimlet class's singleClicked method (described in section 4.1.3)

<singleClicked> Example

```
<singleClicked>
  <canvas type="window" width="300" height="300"/>
  <actionUrl method="get" target=http://maps.google.com/>
</singleClicked>
```

3.5.4 <dragSource> Element

The <dragSource> element specifies a drag source for the Zimlet panel item. A drag source consists of a ZCS object type e.g. a conversation, a contact, or a mail message (for a complete discussion of valid objects and object properties see section 3.5.5) and the action to be performed when a object of that type is dropped on the panel item. There may be zero or more <dragSource> items specified for a Zimlet panel item. The <dragSource> element requires a single attribute named "type". This attribute specifies the drag source type and may be one of the objects specified in section 3.5.5. The <dragSource> item may also contain the following child elements:

PRELIMINARY DRAFT

- `<canvas>` - determines the visual for the drop action. The `<canvas>` element is described in section 3.2.3
- `<actionUrl>` - Specifies the URL to be loaded when an item of the type specified in the `<dragSource>` element's type attribute is dropped onto the Zimlet panel item. The `<actionUrl>` element is described in section 3.2.1. If this element is omitted, then the Zimlet framework will call the Zimlet class' `doDrag` method (described in section 4.1.1) when an object of the type specified by the `<dragSource>` element's type attribute is dragged onto the panel item. If the item is dropped on the panel item, the Zimlet class's `doDrop` method (described in section 4.1.2) is called.

`<dragSource>` Example

```
<dragSource type="Contact">
  <canvas type="inline" width="200" height="150"/>
  <actionUrl target="http://maps.google.com">
    <param name="">${obj.homeAddress}</param>
  </actionUrl>
</dragSource>
```

3.5.5 ZCS Objects and Object Properties

This section describes the set of ZCS objects that may be specified as drag source types. In addition, the set of attributes available as URL parameters is enumerated for each object type.

These public attributes may be accessed by specifying it as the value of a `<actionUrl>` element's `<param>` child. In this case the attribute may be accessed by enclosing it with `"${}"` and scoped with `"obj"`. For example to access the Conversation object's subject attribute, one would specify `${obj.subject}`

The following JavaScript types are converted to URL encoded formats using the following rules:

- **Int** - Passed as simple integer value, e.g. `http://acme.com/service?msgId=8273828`
- **String** - Passed as simple string value, e.g. `http://acme.com/service?subject=Hello%20World`
- **Array** - Passed as a comma separated list of values. If there is a comma in the value, it will be escaped with a backslash: `\,`
- **Boolean** - Passed as 1 for TRUE, and 0 for FALSE
- **Date** - Passed as an integer representing the number of milliseconds since the beginning of the Unix epoch

ZmConv

ZmConv class represents a conversation. A Conversation is a set of email messages related by the same subject and received within a defined period of time. The following attributes are exported for use as URL parameters:

- **id** [Int]- Conversation ID
- **subject** [String] - Conversation subject i.e. subject of the first message in the conversation
- **date** [Date] - Receipt date of the first matched message based on sort order
- **fragment** [String] - Fragment from the first matched message based on sort order
- **participants** [Array(string)] - Up to five participants email addresses including the

PRELIMINARY DRAFT

- originator of the conversation.
- **numMsgs** [Int] - Number of messages in the conversation
- **tags** [Array(Int)] - Union of the set of tags on the messages in this conversation. May be null. Comma separated list when passed to a URL
- **flagged** [Boolean] - True if one or more messages in the conversation has been flagged
- **unread** [Boolean] - True if one or more messages in the conversation is unread
- **attachment** [Boolean] - True if one or messages in the conversation has an attachment
- **forwarded** [Boolean] - True if one or more messages in the conversation has been forwarded
- **sent** [Boolean] - True if the user has participated in the conversation
- **replied** [Boolean] - True if one or more messages have been replied to by the user
- **draft** [Boolean] - True if the conversation contains a draft message. Note that a conversation containing a draft message will only contain that message.

ZmMailMsg

The ZmMailMsg object represents a mail message. The following attributes are exported for use as URL parameters:

- **id** [Int]- Message ID for the message
- **convid** [Int]- Conversation ID for the message
- **from** [String] - Email address from the "From:" header
- **to** [Array(String)] - List of email addresses from the "To:" header
- **cc** [Array(String)] - List of email addresses from the "Cc:" header
- **subject** [String] - Message subject
- **date** [Date] - Receipt or sent date of the message
- **size** [Int] - Size in bytes of the message
- **fragment** [String] - Fragment from the message
- **tags** [Array(Int)] - Set of tags on the message
- **flagged** [Boolean] - True if the message has been flagged
- **unread** [Boolean] - True if the message is unread
- **attachment** [Boolean] - True if the message has an attachment
- **forwarded** [Boolean] - True if the message has been forwarded
- **sent** [Boolean] - True if this message was sent by the user
- **replied** [Boolean] - True if the user has replied to this message
- **draft** [Boolean] - True if this is a draft message.

ZmContact

The ZmContact object represents a contact. The following attributes are exported for use as URL parameters:

- **assistantPhone** [String]
- **callbackPhone** [String]
- **carPhone** [String]
- **company** [String]
- **companyPhone** [String]
- **email** [String]
- **email2** [String]
- **email3** [String]
- **fileAs** [String]
- **firstName** [String]

PRELIMINARY DRAFT

- **homeCity** [String]
- **homeCountry** [String]
- **homeFax** [String]
- **homePhone** [String]
- **homePhone2** [String]
- **homePostalCode** [String]
- **homeState** [String]
- **homeStreet** [String]
- **homeURL** [String]
- **jobTitle** [String]
- **lastName** [String]
- **middleName** [String]
- **mobilePhone** [String]
- **namePrefix** [String]
- **nameSuffix** [String]
- **notes** [String]
- **otherCity** [String]
- **otherCountry** [String]
- **otherFax** [String]
- **otherPhone** [String]
- **otherPostalCode** [String]
- **otherState** [String]
- **otherStreet** [String]
- **otherURL** [String]
- **pager** [String]
- **workCity** [String]
- **workCountry** [String]
- **workFax** [String]
- **workPhone** [String]
- **workPhone2** [String]
- **workPostalCode** [String]
- **workState** [String]
- **workStreet** [String]
- **workURL** [String]

ZmAppointment

- **id** [Int] - Mail item id
- **uid** [String] - iCal UID of the appointment
- **type** [String] - Appointment type: Event or Todo
- **subject** [String] - Appointment subject
- **startDate** [Date] - Appointment start date
- **endDate** [Date] - Appointment end date
- **allDayEvent** [Boolean] - True if an all day event, false otherwise
- **exception** [Boolean] - True if this is an exception to a repeating appointment
- **recurring** [Boolean] - True if this is a recurring appointment
- **alarm** [Boolean] - True if the appointment has an alarm
- **otherAttendees** [Boolean] - True if there are other attendees
- **attendees** [Array(String)] - List of other attendees
- **location** [String] - Location of the appointment
- **notes** [String] - Notes

- **isRecurring** [Boolean] - True if the appointment is a recurring appointment
- **timeZone** [String] - Appointment's timezone

3.6 <userProperties> Element

User properties enable Zimlets to store per user state. If a Zimlet defines user properties, then a properties menu item is added to the Zimlet's panel item and content object context menu. When the user selects the properties menu item, she is presented with a property editor with which to edit the Zimlet's user properties. The property editor is constructed by the Zimlet class' `createPropertyEditor` method (see section 4.3.2). The default implementation of this method will "auto-create" a property editor. Zimlet implementers are free to override this method in order to provide a custom editor.

The <userProperties> element may contain one or more property elements as its children. There are several types of property element. They all share the following set of common attributes:

- **type** - The type of the property such as "string". The set of valid properties is described below
- **name** - The property name
- **label** - The label to be displayed for the property in the property editor. This attribute is only required if the <userProperties> element's editor attribute is set to "standard"
- **visible** - If true, then property is visible in the property editor. This attribute is only required if the <userProperties> element's editor attribute is set to "standard"
- **value** - The default value for the property (optional)

User properties may be accessed within the Zimlet javascript class as follows:

`getUserProperty("PropertyName")` For example, user property *password* would be accessed as: `getUserProperty("password")`. User properties may be accessed within the Zimlet definition file by enclosing the property name with "\${}", and scoping it with "prop". For example to access a property name "username", one would specify `prop.userName`

The available set of property types are described in the following subsections

number type

The number type represents a number. When a user property's type attribute is "number", then the following optional attributes are available:

- **minVal** - The minimum value permitted for the property
- **maxVal** - The maximum value permitted for the property

string type

The string type represents a string. When a user property's type attribute is "string", then the following optional attributes are available:

- **minLength** - The minimum length of the string
- **maxLength** - The maximum length of the string
- **visualType** - May be either "inputField" or "textArea" corresponding to an input field or a multi-line text area respectively. If omitted it defaults to "inputField"
- **numLines** - Applicable only if the visualType is "textArea". Specifies the number of lines that should be displayed in the text area.

password type

The password type represents a password. A password is typically obfuscated as the user types it in. When a user property's type attribute is "password", then the following optional attributes are available:

- minLength - The minimum length of the string
- maxLength - The maximum length of the string

date type

The date type represents a date. Dates are represented internally as milliseconds since the epoch.

boolean type

The Boolean type represents a "true" or "false" value. This is presented to the user as a check button.

enum type

The enum type represents a set of choices. This may be presented to the user as a dropdown box with a list of the valid choices or a set of radio buttons. A property of this type has one or more <item> child elements. The <item> element defines three attributes:

- label - The user interface label for the element. This is a required attribute.
- value - The value for the item. This is a required attribute.
- visualType - This may be either "select" or "radio". If omitted, then this value defaults to "select"

<userProperties> Example

```
<userProperties editor="standard">
  <property type="string" name="uname" label="{msg.uname}"
    visible="true" minLength="4" maxLength="8"/>
  <property type="password" name="pword" label="{msg.pword}"
    visible="true" minLength="4" maxLength="8"/>

  <property type="select" name="pageSize" label="{msg.pageSize}"
    visible="true">
    <item label="10" value="10"/>
    <item label="20" value="20"/>
    <item label="50" value="50"/>
  </property>
</userProperties>
```

4 Zimlet JavaScript Class

The Zimlet framework provides the Zimlet JavaScript base class. Developers wishing to implement Zimlet functionality directly in JavaScript should extend this class. The name of the derived class, as well as the file in which the class is defined should be the same as the Zimlet's name.

The methods comprising the Zimlet base class (ZmZimletBase.js) may be divided into the following categories:

- Methods related to the Zimlet panel items.

PRELIMINARY DRAFT

- Methods related to content objects
- Helper methods

There are certain formal parameters that are common to a number of methods defined by the Zimlet class:

- Parameters common to Zimlet panel item methods:
 - *zmObject* - This parameter is common to certain Zimbra Panel Item methods. It represents the ZCS object that has been dropped onto the panel item. The types of ZCS objects and their respective properties are described in section 3.5.5
- Parameters common to content object methods:
 - *spanElement* - This parameter is common to certain content object methods. It represents the HTML `` element that surrounds the content object text the content that is rendered in the user's browser. The content object methods may manipulate the CSS style of this element to reflect such things as state change based on a user's actions. For example, if a user selects to approve a purchase order from a context menu, the Zimlet developer may wish to turn the content object representing the purchase order green.
 - *contentObjText* - The content object text of the content object being acted upon
- Parameters common across methods:
 - *canvas* - The value of this parameter is driven by the corresponding `<canvas>` element in the Zimlet Definition file. The list below enumerates the value of the *canvas* parameter for each value of the `<canvas>` element's type attribute:
 - **none** - *canvas* will be null
 - **dialog** - *canvas* will be an instance of the `DwtDialog` class. `DwtDialog` is a component of the DHTML Widget Toolkit (DWT) library that is itself a package in the Zimbra AJAX Toolkit. Instances of `DwtDialog` may have DWT widgets such as buttons and trees added to them at runtime. However, the decision to use the DWT widget set is optional. By calling the `getHtmlElement()` method of a composite object a handle may be obtained to composite's DOM element. This element is a `<div>` object that may be arbitrarily manipulated. For example, any HTML content may be placed within this `<div>` by setting its `innerHTML` attribute.
 - **tooltip** - *canvas* will be a `DwtToolTip`
 - **window** - *canvas* is a handle to the new browser window.The only exception to the above is for the `tooltipPoppedUp` method. In this case, the *canvas* parameter will always be a `DwtComposite` object.

The remainder of this section will describe the methods defined by the Zimbra class.

4.1 Zimlet Panel Item Methods

Zimlet panel item methods are invoked via the elements of the `<zimletPanelItem>` element within the Zimlet definition file

4.1.1 doDrag Method

This method is called when an item is dragged on the Zimlet drop target as realized in the UI. It is invoked from within the `<dragSource>` element. This method is only called for the valid types that the Zimlet accepts. This method can perform additional validation based on semantic information beyond the type of the object being dragged onto the Zimlet. This method defines the following formal parameters:

- *zmObject*
- Return true if the drag should be allowed, false otherwise.

4.1.2 doDrop Method

This method is called when an item is dropped on the Zimlet item as realized in the UI. At this point the Zimlet should perform the actions it needs to for the drop. This method defines the following formal parameters:

- *zmObject*
- *canvas*
- Return void

4.1.3 singleClicked Method

This method is called when a panel item is clicked once. Beware that it may be called once or twice during a doubleClick.

- *canvas*
- Return void

4.1.4 doubleClicked Method

This method is called when the Zimlet panel item is double clicked. This method defines the following formal parameters:

- *canvas*
- Return void

4.2 Content Object Methods

Content object methods are invoked via the elements of the <contentObject> element within the Zimlet definition file

4.2.1 match Method

This method is called when content (e.g. a mail message) is being parsed. The match method may be called multiple times for a given piece of content and should apply whatever pattern matching is required to identify objects in the content. This method defines the following formal parameters

- *content* - The content against which to perform a match
- *startIndex* - Index in the content at which to begin the search
- Return the first content object match in the content starting from *startIndex*

4.2.2 clicked Method

The *clicked* method is called when a Zimlet content object is clicked on by the user. This method defines the following formal parameters

- *spanElement*
- *contentObjText*
- *matchContext*
- *canvas*
- Returns void

4.2.3 toolTipPoppedUp Method

This method is called when the tool tip is being popped up. This method defines the following formal parameters:

- *spanElement*
- *contentObjText*
- *matchContext*

- *canvas*
- Returns void

4.2.4 **toolTipPoppedDown Method**

This method is called when the user is popping down a sticky tool tip. It defines the following formal parameters:

- *spanElement*
- *contentObjText*
- *matchContext*
- *canvas*
- Returns null if the tool tip may be popped down, else return a string indicating why the tool tip should not be popped down

4.3 Common Methods

4.3.1 **menuItemSelected Method**

The `menuItemSelected` method is called when a context menu item is selected by the user. It defines the following formal parameters:

- *contextMenu* - Identifies the context menu from which the item was selected. This may be
 - `Zimlet.PANEL_MENU`
 - `Zimlet.CONTENTOBJECT_MENU`.
- *menuItemid* - This is the ID that is provide in the `<menuItem>` elements id attribute
- *spanElement*
- *contentObjText*
- *canvas*
- Return void

4.3.2 **createPropertyEditor Method**

This method is called by the Zimlet framework if there is a `<userProperties>` element specified in the Zimlet definition file. Specifically, the framework will add a properties menu item to the Zimlet panel item and content object action menus (if one or both are defined). When this menu item is selected, the property editor will be presented to the user. This method's responsibility is to create a property editor for the set of properties defined in the `<userProperties>` element. The default implementation of this method will "auto-create" a property editor based on the attributes of the Zimlet's user properties. Zimlet implementers should override this method if a custom property editor is required. `createProperty` defines the following formal parameters:

- *dialog* - An instance of a `DwtDialog` widget. This widget should be populated with the right sets of fields/data to represent the Zimlet's properties. In addition, this method will register a callback for the dialog's "ok" button. This callback will generally save changed properties by calling the `saveUserProperties` method
- Return void

4.3.3 **submitForm Method**

This method is called by the Zimlet framework when the user clicks on the submit button of a form that is generated by the `<formEditor>` element (see 3.2.2). This method defines the following formal parameters:

- *formName* – The name of the form that is being submitted
- *formFields* – a Hash of the form fields. The key is the field name, and the value is the

value the user entered for the field.

4.4 Helper Methods

Helper methods are provided by the Zimlet class to provide common utility functions for derived classes. These methods are typically not overridden.

4.4.1 applyXslt

This method will apply an XSL transformation to an XML document (e.g. one returned from a webservice call). This method defines the following formal parameters:

- *xsltUrl* – This is the URL to the XSLT style sheet.
- *document* – XML document (or AjxXmlDoc) to which the style sheet is to be applied
- Return an AjxXmlDoc representing the transformed document.

4.4.2 checkProperties

Called just before the properties are saved. Allows the Zimlet writer to validate properties one last time before they are saved.

- *props* – The properties that are about to be saved
- Return true if the properties are valid. If a string is returned then an error dialog is displayed.

4.4.3 enableContextMenu Method

This method provides the ability to enable and disable menu items in the context menu. This method takes the following formal parameters:

- *contextMenu* - Identifies the context menu from which the item was selected. This may be
 - Zimlet.PANEL_MENU
 - Zimlet.CONTENTOBJECT_MENU.
- *menuItemid* - The menu item on which to act
- *enabled* - a Boolean that if set to true will enable the menu item, and if set to false will disable the item

4.4.4 getConfigProperty Method

This method returns the value of the requested global or host property. It defines the following formal parameters:

- *propertyName* - This is the name of the property whose value is to be returned. For example: "global.someProperty" or "host.someProperty"
- Return the value of the property, or null if no such property exists

4.4.5 getResource

This method returns the fully qualified URL of a resource specified by the <resource> element.

```
var imageURL = this.getResource("image.gif");
```

4.4.6 getUserPropertyInfo

This method returns a user property's property information. The property information is a hash the content of which depends on the type of the property. All properties support the following keys:

- *name* - The property's name
- *label* - The label to be displayed for the element

PRELIMINARY DRAFT

- *Visible* - True if the property should be displayed in the property editor.
- *Type* - This will be one of:
 - Zimlet.STRING_PROP
 - Zimlet.PASSWORD_PROP
 - Zimlet.NUMBER_PROP
 - Zimlet.DATE_PROP
 - Zimlet.SELECT_PROP

The following is the list of properties supported for the specific property types:

- string
 - *minLen* - Minimum length for the string (optional)
 - *maxLen* - Maximum length for the string (optional)
- password
 - *minLen* - Minimum length for the string (optional)
 - *maxLen* - Maximum length for the string (optional)
- number
 - *minVal* - Minimum value for the number (optional)
 - *maxVal* - Maximum value for the number (optional)
- date
 - N/A
- select
 - *items* - an array of the items in the select. There will always be at least one item in this array. Each item has two keys
 - *label* - The label to be displayed for the item (Required)
 - *value* - The value for the item (Required)

The `getUserPropertyInfo` method defines the following formal parameters:

- *propName* - The user property name
- Returns a has containing the property info, or null if no such property exists

4.4.7 getUsername

This method returns the current user's username

4.4.8 getUserProperty

This method returns the value of the requested user property. It defines the following formal parameters:

- *propertyName* - This is the name of the property whose value is to be returned. For example: "someUserProperty"
- Return the value of the property, or null if no such property exists

4.4.9 saveUserProperties

This method will save the value of any modified properties to ZCS. It takes no formal parameters

4.4.10 sendRequest Method

The `sendRequest` method enables asynchronous communications with a remote server. This method defines the following formal parameters

- *requestStr* - the request string that is to be sent to the remote server
- *serverUrl* - the URL of the remote server. Note that due to security constraints this must be the ZCS server (i.e. the server from which the JavaScript originated); however it is actually possible to communicate with a remote server by using the Zimlet framework's proxy servlet (see section 6)
- *requestHeaders* - a JavaScript array of HTTP request headers
- *callback*. This is an instance of the `AjxCallback` class. The `AjxCallback` class provides a

mechanism for encapsulating callbacks. Its constructor defines the following formal parameters:

- o *obj* - Generally *obj* is a reference to “this” object.
- o *func*- The function to call. If *obj* is not null, then the value of “this” when *func* is called will be *obj*
- o *args* - Arguments to be passed to the *func*. This may be a single value or a JavaScript array of values

An example of how to construct an *AjxCallback* object is as follows:

```
new AjxCallback(this, MyZimlet.prototype.asyncCallback,  
[this._val1, this._val2]);
```

- *useGet* - if True, then the HTTP “get” method is used. In this case the *requestStr* should generally be null since the *serverUrl* will contain the query parameters.

4.4.11 setUserProperty

This method sets the value of a user property. It defines the following formal parameters:

- *propertyName* - This is the name of the property whose value is to be returned. For example: “someUserProperty”
- *value* - The value to which to set the property
- *save* - If true, then the property will be saved (along with any other modified properties)
- Return void
- Throws *ZimletException* if no such property exists or if the value is not valid for the property type. A *ZimletException* is a JavaScript class derived from base *AJAX Exception* class that is a component of the Zimbra *AJAX Toolkit*.

5 Zimlet JSP

A Zimlet JSP implements any server side data retrieval and manipulation for the actions to be performed on the client. The Zimlet JSP code may use any protocol or API needed to fetch data or act on a user’s selection.

For example, for a UPS package tracking Zimlet, a Zimlet JSP implementation may make a call to the UPS’ tracking information API with the matched tracking number in order to get package status and/or delivery information for displaying to the user. The tracking information would be returned to the client for possible display in a context object tool tip.

5.1 Custom Zimbra Tag Library

A Zimlet JSP implementation can include custom tags provided by the Zimbra Tag Library. Such tags can be used to build powerful server side plug-ins based on JSP, which complements the Zimlets by off-loading expensive tasks from the browser, providing better performance, and simplifying client-server communication. Many of the Zimbra resources are made available by Zimbra Tag Library. Refer to *zimbra.tld* for the details.

Available resources:

- message
- conversation
- contact
- note
- property

- zimletconfig

In order to use the tags from Zimbra Tag Library, the tag library description file has to be declared at the top of JSP.

```
<%@ taglib uri="/WEB-INF/zimbra.tld" prefix="z" %>
```

Then the Zimbra custom tags can be used in the JSP.

```

<table border="1">
  <tr>
    <td>id</td>
    <td>subject</td>
    <td>from</td>
    <td>to</td>
    <td>cc</td>
    <td>bcc</td>
  </tr>
<%
String msgid;
String[] msgids = { "488", "489", "405" };
  for (int i = 0; i < msgids.length; i++) {
    msgid = msgids[i];
%>
  <tr>
    <td><%= msgid %></td>
    <td><z:message id='<%= msgid %>' field="subject"/></td>
    <td><z:message id='<%= msgid %>' field="from"/></td>
    <td><z:message id='<%= msgid %>' field="to"/></td>
    <td><z:message id='<%= msgid %>' field="cc"/></td>
    <td><z:message id='<%= msgid %>' field="bcc"/></td>
  </tr>
<%
}
%>
</table>

```

6 Proxy Servlet

An AJAX client running in a web browser is not permitted to directly make requests to servers other than the originating server, as dictated by the browser security control. Instead, the server hosting AJAX client must make proxy requests on behalf of the client. By using the Proxy Servlet, Zimlets can access remote resources from other servers, as well as make requests to 3rd party systems. The default URL binding for Proxy Servlet is /service/proxy. This servlet takes accepts the following parameters:

- target - the target URL
- auth - authentication method (optional). Currently HTTP basic authentication is supported by Proxy Servlet. (auth=basic)
- user - username used for the authentication (optional)
- pass - password used for the authentication (optional)

The Proxy Servlet will copy any data sent through the POST method to the remote server specified by the target parameter. It will also copy any extra non-functional HTTP headers from the request.

The Proxy Servlet checks the target URL against the list of allowed domains that are listed in COS. When the proxy request target does not appear in the allowed domain list, Proxy Servlet will return HTTP error 403 forbidden.

The Proxy Servlet can optionally cache the contents. Only the non-authenticated contents are cached. The cacheable contents are identified by the content-type header. For example, the caching can be turned on only for static images with image/gif or image/jpeg content types. The set of cacheable content types are also listed in COS.

7 Messages Properties Files

The messages properties file consists of a set of Java properties that represent the localizable messages used by the Zimlet. Additional property files should be provided for each desired locale. These files follow the following standard localization convention for property files as follows:

messages_xx_YY.properties

Where xx is the language and YY is the locale. As an example, the properties file for the English language would be named messages_en.properties, and the properties file for the British locale would be named messages_en_GB.properties

The properties files form a hierarchy such that subsequent nodes in the hierarchy need only override messages that need to be changed for the given language or locale.

Message properties may be accessed within the Zimlet definition file as follows: `${msg.propertyName}`. For example, message property `username` would be accessed as follows: `${msg.username}`

8 Zimlet Configuration File

The `config_template.xml` file specifies a template for Zimlet configuration. There are two sections to this file: a section for global configuration, and a section for per host configuration.

The configuration properties may be accessed within the Zimlet definition file as follows::

```
<zimletConfig name="com_acme_Maps" version="1.0">
  <global>
    <property name="defaultToGoogle"></property>
  </global>
  <host name="">
    <property name="googleApi"></property>
    <property name="geocodeUser"></property>
    <property name="geocodePass"></property>
  </host>
</zimletConfig>
```

The properties defined in `config_template.xml` may be accessed within the Zimlet definition file as

follows:

- `#{config.global.propertyName}` - Accesses a global property. For example, the global property `defaultToGoogle` would be accessed as: `#{config.global.defaultToGoogle}`
- `#{config.host.propertyName}` - Access a host specific property. For example, the host property `googleApi` would be accessed by `#{config.host.googleApi}`

Configuration properties are also available to Zimlet JavaScript code via Zimlet classes `getConfigProperty` method (see section 4.4.4 for more information).

9 Zimlet Lifecycle and the Zimlet Management Tool

This section describes the Zimlet lifecycle in conjunction with the Zimlet Management tool (`zmzimletctl`) for managing this lifecycle.

```
zmzimletctl: [command] [ zimlet.zip | config.xml | zimlet ]
  deploy {zimlet.zip} - install, ldapDeploy, grant ACL
                        on default COS, then enable zimlet
  undeploy {zimlet} - remove the zimlet entry from the system
  install {zimlet.zip} - installs the zimlet files on this host
  ldapDeploy {zimlet} - add the zimlet entry to the system
  enable {zimlet} - enables the zimlet
  disable {zimlet} - disables the zimlet
  acl {zimlet} {cos1} grant|deny [{cos2} grant|deny...] - change
                  the ACL for the zimlet on a COS
  listAcls {zimlet} - list ACLs for the Zimlet
  listZimlets - show status of all the Zimlets in the system.
  dumpConfigTemplate {zimlet.zip} - dumps the configuration
  configure {zimlet} {config.xml} - installs the configuration
  listPriority - show the current Zimlet priorities (0 high, 9 low)
  setPriority {zimlet} {priority} - set Zimlet priority
```

9.1 Listing Zimlets

Lists all the Zimlets installed and deployed on the machine, in the LDAP, and enabled on COS.

```
zmzimletctl listZimlets
```

Output:

Installed Zimlet files on this host:

```
com_zimbra_phone
com_zimbra_po
com_zimbra_sforce
com_zimbra_tracking
com_zimbra_url
com_zimbra_ymaps
```

Installed Zimlets in LDAP:

```
com_zimbra_phone
com_zimbra_po (disabled)
com_zimbra_sforce
com_zimbra_tracking
com_zimbra_url
```

```
com_zimbra_ymaps
```

Available Zimlets in COS:

```
default:
  com_zimbra_po
  com_zimbra_sforce
  com_zimbra_tracking
  com_zimbra_url
  com_zimbra_ymaps
```

9.2 Deploying

This is when the Zimlet is first deployed to the ZCS system. Physically this means that a Zimlet may be deployment to a number of host machines running the ZCS. Deployment is the first step in the lifecycle process. The Zimlet may not be in a “runnable” state post deployment.

The command for deploying Zimlets is as follows:

```
zmzimletctl deploy <zimletname>.zip
```

where:

- <zimletname>.zip is the Zimlet bundle

9.3 Configuration

Zimlets may need additional site specific configuration before being ready to run. If such configuration is required, the template for it is provided in the config_template.xml file that is part of the Zimlet bundle. Zimlet configuration may include both global and per host properties. The Zimlet Management Tool provides the functionality for setting up Zimlet configuration. For more information on config_template.xml see section 8.

The commands for configuring Zimlets are as follows:

```
zmzimletctl dumpConfigTemplate <zimletname>.zip
```

where:

- <zimletname>.zip is the Zimlet bundle

The above command will extract the config_template.xml configuration from the Zimlet bundle (if one exists). This file should then be edited for the specific site. For example given the following template:

```
<zimletConfig name="com_acme_Maps" version="1.0">
  <global>
    <property name="defaultToGoogle"></property>
  </global>
  <host name="">
    <property name="googleapi"></property>
    <property name="geocodeuser"></property>
    <property name="geocodepass"></property>
  </host>
</zimletConfig>
```

A site may create the following configuration file:

PRELIMINARY DRAFT

```
<zimletConfig name="com_acme_Maps" version="1.0">
  <global>
    <property name="defaultToGoogle">true</property>
  </global>
  <host name="mailserv1.acme.com">
    <property name="googleapi">uh8ieu54151v48h489uh</property>
    <property name="geocodeuser">Zimbra</property>
    <property name="geocodepass">674g7382a</property>
  </host>
  <host name="mailserv2.acme.com">
    <property name="googleapi">uh7jsh8374uae9js76uh</property>
    <property name="geocodeuser">Zimbra</property>
    <property name="geocodepass">837hs7we8</property>
  </host>
  <host name="mailserv3.acme.com">
    <property name="googleapi">uh8su8e9378sjdg739uh</property>
    <property name="geocodeuser">Zimbra</property>
    <property name="geocodepass">802shd837</property>
  </host>
</zimletConfig>
```

Note that this file may be saved for later modification/distribution. The configuration for Zimlets is set via the following command

```
zmzimletctl configure <configfile.xml>
```

where:

- configfile.xml is the modified template file

The above command will create the appropriate configuration for each host.

9.4 Access Control

By default, Zimlets are deployed such that no access is granted to any user to the Zimlet. Zimlet access control is specified on a Class of Service basis. Thus if a COS is granted access to a given Zimlet, then all members of that COS have access to that Zimlet. The following command permits specifying access control to a Zimlet:

```
zmzimletctl acl <zimletname> (<cosname> grant | deny|)+
```

where:

- <cosname> is the COS to which access is to be granted or denied
- "grant" enables access to the Zimlet for the specified COS, and "deny" denies access

The command for listing Zimlet permission is as follows:

```
zmzimletctl listacls <zimletname>
```

9.5 Priority

Zimlets have a priority. The lower the priority value, the higher priority the Zimlet is. Priority plays an important role for Zimlets that define content objects. Specifically, if there is conflict in the content objects specifications among two Zimlets, then the Zimlet with the higher priority will win. For example, consider a Zimlet that defines a content object representing a URL, and another

PRELIMINARY DRAFT

Zimlet that defines a content object that represents a tracking number. Now assume that at runtime some content has embedded in it a URL that contains what coincidentally passes as a tracking number. If the priority of the Zimlet defining the URL is higher than the Zimlet defining the tracking number, then the “tracking number” that is embedded in the URL would never be highlighted (which may actually be desirable if it is a bogus number - and which emphasizes the point that it is preferable that content objects should be uniquely identifiable)

The following command permits specifying the priority of a Zimlet:

```
zmzimletctl priority <zimletname> <value>
```

Where

- <value> is the priority value with 0 being the highest priority.

If a Zimlet already exists with the specified priority, then the zimlet tool will inform the administrator and prompt him to confirm that he wants to set the priority of the Zimlet to the given value. If the administrator confirms this, then the Zimlet that previously held the requested priority will have its priority incremented (as will all Zimlets of lower priority). If the supplied value is more than one unit greater than the lowest priority Zimlet’s value (i.e. if assigning the priority will result in a “hole” in the priority list), then the next available priority will be assigned. If the priority value is omitted, then the next available lowest priority will be assigned to the Zimlet. By default, when a Zimlet is added to the system, it will assigned the next available lowest priority.

The command for listing Zimlet priorities is as follows:

```
zmzimletctl listpriority
```

9.6 Enabling

Once a Zimlet is configured, it must be enabled. Enabling a Zimlet makes it available to all users to whom access has been granted. The command for enabling a Zimlet is as follows:

```
zmzimletctl enable <zimletname>
```

9.7 Disabling

Disabling a Zimlet makes it unavailable to users. A Zimlet is usually disabled before updating and before undeploying.

```
zmzimletctl disable <zimletname>
```

9.8 Undeploying

At the end of its lifecycle (either because it is being retired or upgraded) a Zimlet is undeployed. The command for undeploying a Zimlet is as follows:

```
zmzimletctl undeploy <zimletname>
```

Where:

When a Zimlet is undeployed from all known hosts, all of its user property, configuration and access control information will be purged from the ZCS.

9.9 Getting Zimlet information

The command for getting zimlet information is as follows:

```
zmzimletctl info <zimletName>+ | all [-acl] [-status] [-vers] [-hosts]
[-a]
```

Where:

- -acl will provide a list of all the COSs which have access to the Zimlet
- -status will indicate the enable/disable state of the Zimlet
- -vers will list the Zimlet's version
- -hosts will list the hosts on which the zimlet is deployed
- -config will output the configuration information (global and per host)
- -a will output all of the above

10 Example <zimlet>.xml: Maps

10.1 com_zimbra_ymaps.xml

```
<zimlet name="com_zimbra_ymaps" version="1.0" description="Yahoo Maps">

  <include>
    http://api.maps.yahoo.com/ajaxymap?v=2.0&amp;appid=ZimbraMail
  </include>
  <include>ymaps.js</include>
  <includeCSS>ymaps.css</includeCSS>

  <resource>blank_pixel.gif</resource>
  <resource>ymaps.gif</resource>

  <handlerObject>Com_Zimbra_YMaps</handlerObject>

  <contentObject type="address">
    <matchOn>
      <regex attrs="ig">
        [\w]{3,}([A-Za-z]\.)*([ \w]*#\d+)?(\r\n|
) [\w]{3,}\x20[A-Za-z]{2}\x20\d{5}(-\d{4})?\b
      </regex>
    </matchOn>
    <onClick>
      <canvas type="window" />
      <actionUrl target="http://maps.yahoo.com/beta/index.php"
        paramStart="#">
        <param name="maxp">search</param>
        <param name="q1">${src.objectContent}</param>
      </actionUrl>
    </onClick>
  </contentObject>

  <zimletPanelItem label="Maps" icon="YMAPS-panelIcon">
    <toolTipText>
      Drag'n'drop a contact to display a Yahoo Map
    </toolTipText>

    <dragSource type="ZmContact" />
  </zimletPanelItem>
</zimlet>
```

PRELIMINARY DRAFT

```
<contextMenu>
  <menuItem label="Visit Yahoo Maps"
    id="MAPS.YAHOO.COM"
    icon="YMAPS-panelIcon">
    <canvas type="window" width="800"
      height="600" />
    <actionUrl target="http://maps.yahoo.com">
      <param name="referrer">
        www.zimbra.com
      </param>
    </actionUrl>
  </menuItem>
</contextMenu>

</zimletPanelItem>
</zimlet>
```