Hotlink Integration

This document explains the secure hotlinks facility by which Agiloft may be integrated with a third party portal.

Overview

Consider the case when a user has logged into the portal and now wants to perform some action in Agiloft such as creating a ticket, checking the status of their existing tickets or updating a ticket.

The user should not have to re-enter another login or password and if the account doesn't exist in Agiloft, it should be created automatically.

The possible invocation sequence may look like this:

1. The portal instructs Agiloft to create a user account if one does not already exist.
2. The portal instructs Agiloft to generate a hotlink for the user to create a ticket.
3. Agiloft generates and returns the hotlink.
4. The portal either displays the hotlink to the user to click or simply redirects the user's browser to that URL.
5. The standard New Ticket form opens and the user can fill in the data.
6. On clicking Finish the record is saved and the user is redirected back to the portal.

The syntax by which the portal makes the requests in steps 1 and 2 is the standard hotlink syntax documented in the Administrator Guide / Access Methods / Hotlinks section of the user manual, with a few extra parameters such as the start and expiration date/time of the hotlink.

Using the expiration time parameter increases security since the hotlink after it has been disclosed to the user will stop working once it has expired. In the scenario when the user clicks on some link displayed in the portal and is immediately redirected to Create Ticket form the expiration time may be relatively short, e.g. 10 seconds from the current time.

The hotlink generated in step 3 is encrypted using a key unique to that knowledgebase so that the user's login and password are never exposed.

The steps 4,5 and 6 of the above mention redirecting to a New Ticket screen, but the Portal can also invoke the New Ticket screen in a popup browser window that could disappear when the user clicks Finish, or can log the user into the full Agiloft GUI.

Example One

The example below uses plain HTML forms to illustrate how to perform the integration tasks. Please be aware that this is done only as a demonstration. The real integration should never use this method as it exposes the admin-level password to the users.

Request Agiloft to generate a hotlink to log a user in.

```
1 <html>
2 <body>
3 <form action="https://localhost/ewws/EWHotlinks" method="post">
4 <input type="hidden" name="$login" value="admin">
5 <input type="hidden" name="$password" value="1">
6 <input type="hidden" name="$lang" value="en">
7 <input type="hidden" name="$KB" value="Demo">
8 <input type="hidden" name="mode" value="debug">
9 <input type="text" name="template" size="200"
10 value="State=Main&user=guest&project=Demo
11 &expiration={to_number(now()+5minute)}
12 &start={to_number(now())}&exiturl={defaultExitURL()}">
13 <input type="submit" value="Go">
14 </form>
15 </body>
16 </html>
```

Line 3 contains the URL of hotlink generator entry point in Agiloft. To run this sample on a machine other than your default Agiloft server, replace "localhost" with the desired server's IP or host name. You may also need to specify the port if running on a port other than 80.

Lines 4-7 contain mandatory parameters for hotlink generation that should always be present. These are: the login /password for an admin-level user in Agiloft, your KnowledgeBase name, and the default language.

The parameter in line 8 is optional, and has two possible values: "debug" and "javascript". If left unspecified, the default is "debug". When "debug" is used, the hotlink generator returns the result as text/plain file in Javascript notation. In your real implementation you may find "javascript" more suitable as this returns a text/html snippet with a "script" tag that makes a client-redirect via Javascript to the URL generated. This snippet can be inserted directly into the stream being returned to the user's browser.

Lines 9-12 contains the hotlink template, which tells Agiloft what should be in the final hotlink to be generated and encrypted.

Template parameter

```
State=Main&user=guest&project=Demo&expiration={to_number(now()+5minute)}
&start={to_number(now())}&exiturl={defaultExitURL()}
```

This follows the general Agiloft hotlinks notation with some values being specified as formulas to be evaluated at the time of call.

Please note the absence of the user password. It is assumed that the call to generate the hotlink is coming from a trusted channel e.g. portal's backend and by the time the link will be displayed/disclosed to the user the portal has performed the authentication of the user.

Constant values

In the beginning of the template we can see simple constants known to the portal at the time of call:

State=Main - takes the user to the Main interface on login.

user=guest - this is the name of the user profile we are using to log in to Agiloft. Can be derived from the portal user's login in the real integration scenario.

project=Demo - the name of the KnowledgeBase you're logging in to. Please note that this should match the *$KB* parameter value above.

Formulas

The formulas (text, enclosed by {}) are used to define values to be calculated by Agiloft at the time of call.

To generate a hotlink, you need to convey the hotlink activation time via **start** parameter, and for security reasons, you should define the **expiration** time.

Using the function *now()* with a time offset of your choice is a preferred way to perform this. Additionally function *to_number* has to be applied to the results of *now()*.

The function now() calculates time with respect to the timezone configured for the Agiloft knowledgebase. In general this does not have to match the user or portal timezone.

In the example the hotlink is requested to be activated immediately (*start={to_number(now())}*) and to be valid for the next five minutes only (*expiration={to_number(now()+5minute)}*).

You can use the value of -1 for the expiration time if you don't want the hotlink to expire. Obviously, this is not recommended for security reasons.

The last parameter in the template is the **exiturl**. Two functions are provided by Agiloft - defaultExitURL() and defaultCancelURL(), that pull the global variable URLs from the your installation of Agiloft. You can also hard-code any pre-set URL as a value there.

Encryption keys

Before testing your hotlinks, you should check if Agiloft contains valid hotlink encryption keys. Login as admin-level user, go to Access/Automatic login hotlinks. Check value "Keys generated at:", if they are not generated, press "Generate". Using this encryption key allows you to hide the variables which might be a security problem if sent in plain text.

The result of the call in "debug" mode would be similar to:

```
var redirectURL=
'https://localhost/gui2/login.jsp?genhotlink=
X7g7z+QhGmQAailWCwVumrj7gjF9jTD0MeyB1ZK/3DUZ5VYM8ejftln0r8x+zv6ZDoJR+PV7
v33Bz4ehE4MkyuaOfaDZv0dZt38dJIJdzmX38dHwEz6WVg==&genproject=Demo'
```

The result of the call in "javascript" mode would be similar to:

```
<script>
location.href=
'https://support.enterprisewizard.com/gui2/login.jsp?genhotlink=
X7g7z+QhGmQAailWCwVumrj7gjF9jTD0MeyB1ZK/3DUZ5VYM8ejftln0r8x+zv6ZDoJR+PV7
v33Bz4ehE4MkyuaOfaDZv0dZt38dJIJdzmX38dHwEz6WVg==&genproject=Demo'
</script>
```

Example Two

Editing an existing record when the ID is known by the portal. The example below uses plain HTML forms to illustrate how to perform the integration tasks. Please be aware that this is done **only** as a demonstration. The real integration should **never** use this method as it exposes the admin-level password to the users.

This is a more advanced case, because it requires more data than just the login/password information from the portal side to generate the hotlink. For this case, the portal must pass to the generator:

- The ID of the record.
- The Agiloft logical table name.
- The subtable name.

The table and subtable names are explicitly defined since they cannot be changed by anyone but admin-level users.

One can look these names up in Agiloft by navigating to **Setup > Tables > Edit > General tab** and store them in some configuration.

```
<html>
<body>
<form action="http://192.168.1.4:8080/ewws/EWHotlinks" method="post">
<input type="hidden" name="$login" value="admin">
<input type="hidden" name="$password" value="qwerty">
<input type="hidden" name="$lang" value="en">
<input type="hidden" name="$KB" value="Demo">
<input type="hidden" name="mode" value="javascript">
<input type="hidden" name="tablename" value="case">
<input type="hidden" name="subtablename" value="case">
<input type="text" name="template" size="200"
value="State=Edit:{$tablename}.{$subtablename}&record={$recordid}
&user=guest&project=Demo&expiration={to_number(now()+5minute)}
&start={to_number(now())}&exiturl={defaultExitURL()}
&record_access=edit"><br>
Record ID:<input type="text" name="recordid"><br>
<input type="submit" value="Go">
</form>
</body>
</html>
```

Note that the ID of the record, table and subtable names are passed as HTTP parameters, with corresponding formulas referencing them by name. These can also be hardcoded directly in the hotlink template instead. The user "guest" used for demonstration usually doesn't have access to the power user interface and no permissions to edit other user's records. Use a different user in your tests.

Example Three

The example below uses plain HTML forms to illustrate how to perform the integration tasks. Please be aware that this is done only as a demonstration. The real integration should never use this method as it exposes the admin-level password to the users. Create a user in Agiloft

```
<html>
<body>
<form action="https://localhost/ewws/EWHotlinks" method="post">
<input type="hidden" name="$login" value="admin">
<input type="hidden" name="$password" value="qwerty">
<input type="hidden" name="$lang" value="en">
<input type="hidden" name="$KB" value="Demo">
<input type="hidden" name="mode" value="javascript">
<input type="hidden" name="tablename" value="contacts">
<input type="hidden" name="subtablename" value="customer">
<input type="text" name="template" size="200"
value="State=New:{$tablename}.{$subtablename}&user=guest&project=Demo
&expiration={to_number(now()+5minute)}&start={to_number(now())}
&exiturl={defaultExitURL()}&gui=no"><br>
<input type="submit" value="Go">
</form>
</body>
</html>
```

The changes are limited to the table/subtable name, and some parts of the hotlink template, such as State=New instead of Edit and gui=no to log the user out after the entry is created.

Example Four

The example below uses plain HTML forms to illustrate how to perform the integration tasks. Please be aware that this is done **only** as a demonstration. The real integration should **never** use this method as it exposes the admin-level password to the users. Update the user information in Agiloft.

This is actually a combination of the above:

```
<html>
<body>
<form action="localhost/ewws/EWHotlinks" method="post">
<input type="hidden" name="$login" value="admin">
<input type="hidden" name="$password" value="qwerty">
<input type="hidden" name="$lang" value="en">
<input type="hidden" name="$KB" value="Demo">
<input type="hidden" name="mode" value="javascript">
<input type="hidden" name="tablename" value="contacts">
<input type="hidden" name="subtablename" value="customer">
<input type="text" name="template" size="200"
value="State=Edit:{$tablename}.{$subtablename}&record={$recordid}
&user=guest
&project=Demo&expiration={to_number(now()+5minute)}
&start={to_number(now())}
&exiturl={defaultExitURL()}&record_access=edit"><br>
Record ID:<input type="text" name="recordid"><br>
<input type="submit" value="Go">
</form>
</body>
</html>
```

To be able to edit the user information, the portal must know the user's ID. REST API may be used for that.

Please note that for EndUsers the permissions to edit records in the "Contacts" table are usually given for their own record only, so the ID of the record to update must match the login in "template" parameter with user=guest in the example.