Print Anything

Customization & Enhancement Guide

Using This Document

This document provides information about the installation and customisation of Print Anything “packages” and links to invoke them.

This document is intended for Salesforce administrators. Ideally you will also have some knowledge of Appexchange API queries (SOQL & retrieve) and use of the Excel Connector for Salesforce.

Table of Contents

1. Tutorial : creating a new Printer Package

What: **Create a new package that can be run and display data**

Why: Allows an administrator to create a new Printable view and invoke it. This provides an understanding of all of the features of the Print Anything tool.

How: For this exercise, we'll build a Printable view of the event Record. It will represent a paper based invitation for the event. It will include details from the Contact, the Account and a list of the other Contacts from that Account.

This exercise will take approximately 15 minutes.

Step 1 : Create a new Printer Package record. Name the package “EventInvitation”. You will not be allowed to use whitespace in the name and it must be unique. This is because the name is used to select the package at runtime and allows it to be invoked in multiple Salesforce instances without changing ids.

Leave the other package fields empty for now. We will return to them later.

Step 2 : Now we will create a button on the Event detail page that will invoke the package. The general structure of the url for the printer is...

```
{!URLFOR(
$SControl.PRINTANY__PrintAnythingDriver,
Event.Id,
[packageId="EventInvitation", debug="1"])
}
```

You can see that the driver is identified by name, as is the package. This provides the portability.

Name the button “Print Invitation”, set it to Display in a new window and the Content Source is URL. You will also need to edit the Event page layout to include this new button on the detail page.

Once you have created this button, you can invoke the printer from any event. Make sure that the event has a Contact associated with it. This Contact should be connected to an Account with more than one Contact.

To test, create a meeting for your Contact and then click the button on the event detail page.

You'll see it add the parameters as data and then tell you that there are no templates available.

<u>Notes</u>

You can remove the debug=1 parameter to see how it will run without all the debug/development output

As you have probably realized, you can add any parameter you like to these urls, the driver will parse them and add them as merge fields.

Step 3 : Now we'll add the first query to the package. In this example, it is getting all of the details for the event record. Since we know the id of the record, we can use the retrieve call instead of a SOQL query.

Navigate to your new Printer Package and create a new query from the related list. Populate it with these values

Field	***Value***	***What this does?***
Name	Event	This label is displayed to the user as the query is being executed
Sequence	1	Controls the where in the sequence of queries, this query is executed
Retrieve Entity	Event	The API name of the entity being queried
Retrieve Field	Parameter.eventId	Filters the rows from the entity above. Can be a retrieve or a SOQL where condition. More information is coming in this tutorial
Singleton Field List	Id	There are many situations where queries might return more than one record. In this case, the driver can display a selector for the user. The fields displayed by this selector are listed here, in comma delimited format. Note : be careful to avoid spaces in this list of fields. In this example, we use a dummy field to tell that driver that one record is expected from the retrieve, even though we know that only one record exists. This is because the data label for single row queries is different for multi-row queries.

Once you have added this query, try running the printer once again. You can do this by clicking the button or refreshing the page. The best way is to click the "Reload Package" link in the debug header. This saves you a lot of time as you change your package so try using two browser windows, one for the package and one for the printable view.

Step 4 : Add a query for the Contact and Account. Add a query to your package as follows

Field	***Value***	***What this does?***
Name	Contact and Account	
Sequence	2	
SOQL	select c1.Id, c1.FirstName,c1.LastName,c1.Fax, c1.Email,c1.Phone, c1.Account.Id,c1.Account.Name from Contact c1 where c1.Id = '{Event1.WhoId}'	The query to be run. You can use SOQL or retrieve but not both, a validation rule will stop you from populating both fields.
Singleton Field List	Id	For this query we expect one Contact row to be returned so we use the same dummy value to tell the driver to expect one record. The related Account data will automatically be added as merge data.

Now reload the package to see them run.

An important thing to notice is that you can use any SOQL for a query. Each query is merged before execution, allowing you to use the results from previous queries in your SOQL. A good tool for building your SOQL queries is the "Apex Explorer" https://wiki.apexdevnet.com/index.php/Tools

Also note that the sequence for each query is appended to the Entity name, allowing an entity to be queried more than once without a naming collision in the merge data.

This query is an example of a SOQL relationship query which joins to a parent (i.e the Account) record.

Step 5 : Adding a multi-row SOQL query. Now we want to get data for all the rest of the Contacts at this Account. Since we don't know their ids, we cannot use retrieve so we will use SOQL.

Name	*Sequence*	*SOQL*
Account and Other Contacts	3	select a.Name, a.Phone, (select c.Id, c.FirstName, c.LastName, c.Email, c.Title from a.Contacts c where c.Id <> '{Contact2.Id}') from Account a where a.Id = '{Contact2.Account.Id}'

Reload the package once again.

This is an example of a relationship query with a child subquery. You can see that the two sets of records are added as merge data when subqueries are used.

Step 6 : Adding a multi-row retrieve. Where you have a set of ids from a previous query, it is possible to use the retrieve call to fetch them.

Name	*Sequence*	*Retrieve Entity*	*Retrieve Field*	*Singleton Field List*
Contacts Again	4	Contact	Account3.Contacts.Id.n	

Now reload to see the extra query. Notice that it has retrieved all of the Contacts from a previous multi-record query. In this example, this is not a particularly useful query but there are cases where this is useful e.g. retrieving Products associated with Opportunities via the OpportunityLineItem entity.

You can remove this query for this tutorial since it does not provide any extra data used in the template.

Step 7 : Adding a template. Now we want to add a merge template for all this data to be displayed. Create a new “Template” from the related list in your Package. Use the following values :

Name : English A4
Default Date format : dd/MM/yy
Content :

```
<center>

 <table border="0" width=90%>

  <tr>
```

```
    <td valign=top>
     <table>
     <tr><td class="tutorialText" nowrap><b>Date :</b></td><td class="tutorialText">
{DATE:Today}</td></tr>
     <p></p>
     <tr><td class="tutorialText" nowrap><b>Phone:</b></td>
     <td class="tutorialText">{Contact2.Phone}</td></tr>
     <tr><td class="tutorialText" nowrap><b>Client Fax:</b></td>
     <td class="tutorialText" nowrap>{Contact2.Fax}</td></tr>
     <tr><td class="tutorialText" nowrap><b>Client Email:</b></td>
     <td class="tutorialText">{Contact2.Email}</td></tr>
     <tr><td class="tutorialText" nowrap><b>Company:</b></td>
     <td class="tutorialText">{Account3.Name}</td></tr>
     <tr><td class="tutorialText" nowrap><b>Contact Email:</b></td>
     <td class="tutorialText">{UserInfo.Email}</td></tr>
     </table>
    </td>
   </tr>

   <tr><td height="20" bgcolor="#FFFFFF" colspan=4/></tr>

   <tr><td colspan=4>

    <p class="tutorialText">Dear {Contact2.FirstName},</p>
    <p class="tutorialText">Thank you for your interest.</p>
    <p class="tutorialText">Re-confirming our appointment, we are meeting at {Event1.Location} on
{DATE[EE, MMM yyyy hh:mm a]:Event1.ActivityDateTime}</p>
    <p class="tutorialText">Here is a list of your other colleagues that we have met before:</p>

    <table border=1>
    <tr><th>First Name</th><th>Last Name</th><th>Title</th></tr>
    <prtany:repeat record="Account3.Contacts">
     <tr>
      <td class="tutorialText">{Account3.Contacts.FirstName.n}</td>
      <td class="tutorialText">{Account3.Contacts.LastName.n}</td>
      <td class="tutorialText">{Account3.Contacts.Title.n}</td>
      <prtany:notEmpty field="Account3.Contacts.Email.n">
       <td class="tutorialText">Email : {Account3.Contacts.Email.n}</td>
      </prtany:notEmpty>
     </tr>
    </prtany:repeat>
    </table>

    <p class="tutorialText">Thanks, {UserInfo.FullName}</p>
    </td></tr>

  </table>

</center>
```

Now reload the package and you'll see the template detected and use for display. There are a number of features to notice :

1. Merge fields are added using ***{entity<seq>.field}*** syntax. You can see the fields available in debug mode.
2. For a date field, you can use ***{DATE:entity.field}*** to invoke the date formatter. Although there is no example in this template, you can also use ***{CURRENCY:entity.field}*** where numbers need to be displayed as monetary values.
3. When you want a date format that is not the default, use the ***{DATE[format]:entity.field}*** syntax. In the content above, you can see the meeting date & time being formatted this way, while the date at the top of the page uses the template default date format.
4. For multi-row queries, anything in between ***<prtany:repeat record="entity">*** and ***</prtany:repeat>*** will be repeated in a loop. When looping, you can refer to the individual records using ***{entity.field.n}***
5. Conditional display of html is possible using ***<prtany:notEmpty field="Account3.Contacts.Email.n">*** and ***</prtany:notEmpty>***. If a value exists for the merge field then the html is displayed. This is useful when displaying a percentage (e.g. 50%) or currency (US$50) and displaying correctly when no value is present. It can also be used to have templates use custom fields that may not exist in an org e.g. in the Quote printer, the product image id custom field is only used if it exists and is populated.

Step 8: Styling your fonts i.e. applying CSS. To make a printable view look professional, it is necessary to specify fonts and this is best done using CSS. Print Anything has a utility method to apply this after the merge. Add the following two lines to the CSS_Javascript field of your template.

```
setStyleByClass('td','tutorialText','font','bold 10px Lucida Grande, Geneva, Verdana, Arial, Helvetica, sans-serif');
setStyleByClass('p','tutorialText','font','bold 10px Lucida Grande, Geneva, Verdana, Arial, Helvetica, sans-serif');
```

Now reload the template and you'll see the fonts applied to the text. Note how the classes match the classes defined in the html.

Note : the CSS attributes passed to this function are javascript DOM attributes, not CSS attributes that you use in a CSS stylesheet. For example, where you would use background-color in CSS, you use backgroundColor in javascript DOM calls.

This template field is mostly used for applying CSS but you can add any javascript to be executed after the merge e.g. manipulating the DOM model. There is also a pre-merge field in the package record. This is for adding javascript to change the merge data available if you need to and are capable of writing javascript.

Step 9 : Creating more than one template. There are many circumstances where you need more than one template for the same data e.g. two different languages or two different print formats for envelopes with varied address windows. To support this, it is possible to have more than one template available in a package. When the printer runs, the last step will prompt the user to choose the template.

Clone the "English A4" template and call the new record "French A4". There is no need to translate the template although in a real example, this is exactly what you would do.

Now reload the package (note : not the template but the package) and you'll see the prompt for the choice of templates.

Step 10 : Automatic Template Selection

In many cases you can use data from your queries to know which template you wish to use e.g. a language custom field from a Contact record. Configuring this is very simple. Find the field you wish to use from the debug footer and copy its field name. Paste this value into the ***Template Selection***

Field of the package. Now copy the value of this field and paste it into the ***Selection Value*** field of the English template. Now reload the package and, instead of seeing the template selector, you should see the matched template being automatically selected.

Step 11 : Editing templates using the Field Watcher tool. Now that we have a working package, it is time to make changes to the templates. The edit page for Salesforce records is not very easy to use when editing html content inside a text area field. It is much easier to edit the content in a file on your PC and have those changes saved back to the template record.

Of course you can do this yourself by Cut/Paste but there is a better way. Using the "Field Watcher" utility (www.buikhuizen.com/watcher) , you can work on a copy of the Content field for your template and each time you save changes to the local file, the watcher will save those changes back to your template record in Salesforce. This means you can make a change and then "Reload template" to see the changes in real time.

Step 12 : Templates larger than 32k. When creating complex views, the templates can quickly grow to be larger than 32k which is the maximum size for a Salesforce long text field. In this case you can use a document to store the template html. To use a document, do the following…

1. Create an html document on your desktop called EngA4.html and paste in your html content
2. Create a document in Salesforce using the file on your desktop
3. Copy the document id (the 15 character id in the browser address field) and paste in into the ***Content Document*** field of the template
4. Reload the template to test

As above in Step 11, you can use a "Document Watcher" which is another utility from the same location as the "Field Watcher" above.
Important Note : When using documents as a template source you must pay attention to the file encoding. If your document contains CJK i.e multi-byte characters then make sure that your file is saved using UTF-16 encoding. This allows for multibyte character support and it is the encoding scheme expected by the driver. No other encoding will work in all browsers for CJK characters.

Note : The CSS_Javascript field is still used when using documents as your template source.

2. Installing pre-built packages from Excel

What:	**Create templates that other people have created**
Why:	Allows the sharing & re-use of templates developed by other Print Anything users.
	This assumes that you have the Salesforce connector for Excel (http://sforce.sourceforge.net) installed and that you know how to use it.
How:	All of the steps in the tutorial above may be created using the worksheet called Package-EventInvite.xls which exists in the same folder as this document.
	Open the worksheet in Excel and then work through each sheet, from left to right, using the “insert” function to create the records that make up the package. NOTE : Sheet 4 is not an insert but is a query using the “Query Table Data” command of the Salesforce Excel connector.
	There is another file called Package-QuotePrinter.xls which creates a package to print Quotes. This package depends on the Appexchange Sales Quote application i.e. install the Quoting application first.
	If you have created a useful package, use this worksheet as a way to share your work in a way which is consistent for other users.