Useful functions for developers to use instead of creating alternate reports in Microsoft Dynamics GP

CONFIDENTIAL ARTICLE

(The information in this article is provided to you in accordance with your Confidentiality Agreement)

Partner Only Article
Article ID : 888884
Last Review : N/A
Revision : 4.0

INTRODUCTION

Third-party integrations can now use function triggers to return third-party data to Report Writer reports in Dexterity in Microsoft Dynamics GP and in Microsoft Business Solutions - Great Plains 8.0. Developers can use these triggers instead of creating alternate reports.

MORE INFORMATION

Developers who write integrations to Microsoft Dynamics GP windows store their data in tables that are parallel to the Microsoft Dynamics GP table. The new fields are frequently added to the Microsoft Dynamics GP window or to a new window that sits next to the original window. To print the data, the user can add the parallel table to an existing report. Or, more frequently, the developer creates Report Writer functions to return the custom data to the report. In either case, the report has to be made an alternate report to the third-party product.

Unfortunately, sometimes large customizations have already been made to the report, and those changes must be recreated on the alternate report in the third-party reports dictionary. To help resolve this problem, six new functions have been introduced specifically to be used to create a third-party integration point with Microsoft Dynamics GP 9.0 Dexterity and with Report Writer.

These scripts are designed to provide generic function calls in Report Writer that the third party can trigger on to return the requested data. When the developer uses these scripts, no alternate report is required. Additionally, the customer can continue to use the previously customized report.

The new Report Writer functions are primarily intended to return either string data or currency data for parallel tables that integrate with Sales Order Processing (SOP) or with Purchase Order Processing (POP). However, these functions may be used for any type of data. The parameters of the functions have predefined uses. However, the developer may also use these parameters for other purposes.

The rw_TableHeaderCurrency function

```
function returns currency sData;

in integer dict_id;          {Dictionary ID}
in string report_name;               {Report Name}
in string sNumber;           {control field}
in integer sType;            {control field}
in integer iControl;                 {which piece of data to return}
```

This function returns a currency amount or integer data to the calculated field.

The first parameter is the dictionary ID of the third-party product that the calculated field must use. The intention is that the third-party product will validate the dict_id parameter against its own dictionary ID and then exit if the function does not correspond to this product.

The second parameter is the name of the report that this function was called from. This parameter may not be used in all situations, but the report name is passed to the function if it is required.

The third parameter is a string that is part of the control field. For the typical SOP or POP integration, this is the **SOP Number** or the **PO Number** field.

The fourth parameter is an integer that is part of the control field. For the typical SOP or POP integration, this is the **SOP Type** or the **PO Type** field.

The fifth parameter is an integer that is used to determine which piece of data to return. Typically, an integrating application has several new fields that correspond to a Microsoft Dynamics GP table. Because the Report Writer functions can return only one piece of data at a time, retrieving three separate fields requires three calculated fields to call the same function three times. The **iControl** parameter tells the function which field to return.

The rw_TableHeaderString function

```
function returns string sData;

in integer dict_id;          {Dictionary ID}
in string report_name;               {Report Name}
in string sNumber;           {control field}
in integer sType;            {control field}
in integer iControl;                 {which piece of data to return}
```

This function is the same as the **rw TableHeaderCurrency** function. However, it returns a string instead of a currency field.

The rw_TableLineCurrency function

```
function returns currency sData;

in integer dict_id;                  {Dictionary ID}
in string report_name;                       {Report Name}
in string sNumber;                   {control field 1}
in integer sType;                    {control field 2}
in currency cSequenceOne;            {control field 3}
in currency cSequenceTwo;            {control field 4}
in integer iControl;                         {which piece of data to get}
```

This function returns a currency amount or integer data to the calculated field that it is called from.

The first four parameters are the same as in the **rw_TableHeaderCurrency**function.

The fifth parameter is a currency field that is part of the control field. For the typical SOP integration, this is the **Component Sequence** field. For the typical POP integration, this is the **Break Field 1** field.

The sixth parameter is a currency field that is part of the control field. For the typical SOP integration, this field is the **Line Item Sequence** field. For the typical POP integration, this field remains empty because the **POP_POLine** table only has three key segments.

The seventh parameter is the same as in the **rw_TableHeaderCurrency** function.

The rw_TableLineString function

```
function returns string sData;

in integer dict_id;                  {Dictionary ID}
in string report_name;                       {Report Name}
in string sNumber;                   {control field 1}
in integer sType;                    {control field 2}
in currency cSequenceOne;            {control field 3}
in currency cSequenceTwo;            {control field 4}
in integer iControl;                         {which piece of data to get}
```

This function is the same as the **rw_TableLineCurrency** function. However, it returns a string instead of a currency field.

The rw_ReportStart function

```
function returns string Status;

in integer dict_id;          {Dictionary ID}
in string report_name;               {Report Name}
```

This function can be called from a calculated field in a **Report Header** section. This function can be used to notify the third-party application that the report has started.

The rw_ReportEnd function

```
function returns string Status;

in integer dict_id;          {Dictionary ID}
in string report_name;               {Report Name}
```

This function can be called from a calculated field in a Report Footer section. This function can be used to notify the third-party application that the report has ended.

Consider the following scenario:

You create an SOP_Ship_Weight table that is parallel to the Microsoft Dynamics GP SOP_LINE_WORK table. This new table stores the Item Shipping Weight element and the Vendor ID element. The **Item Shipping Weight** field uses the integer 1 to determine the field data that is to be returned. The **Vendor ID** field uses the integer 2 to determine the field data that is to be returned. The primary key structure for the SOP_Ship_Weight table is the same primary key structure for the SOP_LINE_WORK table.

Add data from the SOP_Ship_Weight table to the SOP Blank Invoice report. You should use triggers in Report Writer functions to retrieve data from the SOP_Ship_Weight table to use in the report.

In this scenario, the user performs the following steps on the Microsoft Dynamics GP client computer:

1. On the client computer, start Report Writer in Microsoft Dynamics GP, and then open the SOP Blank Invoice report.
2. Add a new calculated field, and then name the field **LineSequence**. In the **Result Type** list, select **Currency**. Insert the **Line Item Sequence** field from the SOP Line Work table into the **Expression** field. Click **OK** to save the field.

   Note The purpose of this calculated field is to set the **Line Item Sequence** field as a currency field so that you can pass it to the **Report Writer** function.
3. Repeat step 2 for the **Component Sequence** field in the SOP Line Work table. Name the new field **ComponentSequence**.
4. Add a calculated field on the report, select the user-defined function that is called **rw_TableLineString**, and then click **Add**.
   a. For the first parameter, select **Integer** in the **Type** drop-down list, enter the product ID for the product, and then click **Add**. Alternatively, you can insert a constant of **0**.
   b. For the second parameter, select **String** in the **Type** drop-down list on the **Constants** tab, type **Invoice** in the **Constant** field, and then click **Add**.
   c. For the third parameter, select the **SOP Number** field in the **SOP Line Work** table, and then click **Add**.
   d. For the fourth parameter, select the **SOP Type** field in the **SOP Line Work** table, and then click **Add**.
   e. For the fifth parameter, select the **LineSequence** calculated field that was created in step 2, and then click **Add**.
   f. For the sixth parameter, select the **ComponentSequence** calculated field that was created in step 3, and then click **Add**.
   g. For the last parameter, select **Integer** in the **Type** drop-down list, enter a constant of 1, and then click **Add**. Click **OK** to save the calculated field. This field retrieves the Item Shipping Weight data from the SOP_Ship_Weight table.

   Note Step 4 creates a calculated field that calls the **rw_TableLineString** function. The **rw_TableLineString** function exists in the base Microsoft Dynamics GP dictionary and does not do anything. It is a function that contains parameters. It is present in the Microsoft Dynamics GP dictionary so that developers can attach a trigger to that function.
5. Repeat steps 1 through 4 in another calculated field. However, for the last parameter, select **Integer** in the **Type** drop-down list, enter a constant of **2**, and then click **Add**. Click **OK** to save the calculated field. This calculated field retrieves the Vendor ID data from the SOP_Ship_Weight table.

The calculated field on the report calls the **rw_TableLineString** function and passes in the parameters. The **rw_TableLineString** function is run in Microsoft Dynamics GP. However, it is a shell. Therefore, the function runs, but it does not do anything. When the function runs in Microsoft Dynamics GP, the trigger in the third-party application occurs. You retrieve the data in the trigger processing function, and you pass the data back in the function returns parameter. That data is the data that prints on the report.

The last steps are completed by the developer in Dexterity in Microsoft Dynamics GP. The developer puts a function trigger in the third-party application in the **rw_TableLineString** function. Then, in the trigger-handling script, the developer retrieves the corresponding record, determines the data to be returned, and then returns the formatted data. This process is shown in the following sample code.

```
{Startup script for trigger registration}

local integer l_result;

l_result = Trigger_RegisterFunction(function rw_TableLineString, TRIGGER_AFTER_ORIGINAL, function gpTriggerTableLineString);
if l_result <> SY_NOERR then
    warning "The rw_TableLineString trigger is not registered.";
end if;


{function gpTriggerTableLineString to return the data}
function returns string sData;

in integer dict_id;          {Dictionary ID}
```

```
in string report_name;                {Report Name}
in string sNumber;           {control field 1}
in integer sType;            {control field 2}
in currency cSequenceOne;    {control field 3}
in currency cSequenceTwo;    {control field 4}
in integer iControl;                  {which piece of data to get}

{If this function is not for our product then quit}
if dict_id <> Runtime_GetCurrentProductID() then
          abort script;
end if;

{The SOP_Ship_Weight table is a parallel table to the SOP_LINE_WORK
table and has the same primary key structure as that table: SOP Number,
SOP Type, Component Sequence, Line Item Sequence.  The report_name parameter
is not referenced here as we want to return the field data regardless of which
report it comes from.}

'SOP Number' of table SOP_Ship_Weight = sNumber;
'SOP Type' of table SOP_Ship_Weight = sType;
'Component Sequence' of table SOP_Ship_Weight = cSequenceOne;
'Line Item Sequence' of table SOP_Ship_Weight = cSequenceTwo;
get table SOP_Ship_Weight;

if err() = OKAY then
          {There are two additional pieces of data in this table,
          Item Shipping Weight, and 'Vendor ID'.  Integer values 1 and 2 will be used
          respectively to identify the fields}
          case iControl
                    in [1]
                              {Item Shipping Weight is stored as an integer.  Divide by
                              100.0 to force decimal places and then convert to a string}
                              sData= str('Item Shipping Weight' of table SOP_Ship_Weight /100.0);
                    in [2]
                              sData = 'Vendor ID' of table SOP_Ship_Weight;
          end case;

else
          clear sData;
end if;
```

APPLIES TO

- Dexterity, when used with:
  Microsoft Dynamics GP 10.0
  Microsoft Dynamics GP 9.0
  Microsoft Business Solutions–Great Plains 8.0

Keywords: kbmbsmigrate kbmbspartner kbinfo KB888884