

Technical Reference



How to add a Collection Management Note – via Dexterity.

Call `AddCollectionsManagementNote`, with these parameters:

in	string	CustomerNumber;	
in	text	NoteText;	
out	boolean	NoteSuccessfullyAdded;	
inout	date	ContactDate;	{if empty, will be sysdate}
inout	time	ContactTime;	{if empty, will be systime}
inout	string	UserWhoCreatedNote;	{if empty, current user}
	date	NoteCreationDate;	{if empty, sysdate}
inout	time	NoteCreationTime;	{if empty, systime}
inout	string	ActionPromisedID;	{can be empty}
inout	date	ActionDate;	{if empty, and ActionPromisedID exists then sysdate+FollowUpActionDays from action}
inout	string	ActionAssignedTo;	{if empty, then User ID of globals}
inout	Boolean	ActionCompleted;	{if ActionPromisedID, then from the action, else false}
inout	date	ActionCompletedDate;	{if empty, and ActionPromisedID and ActionCompleted=true then sysdate}
inout	time	ActionCompletedTime;	{if empty, and ActionPromisedID and ActionCompleted=true then systime}
inout	currency	AmountPromised;	inout currency AmountReceived;
inout	string	NoteDisplayString;	{if empty, will be first 70 character of the NoteText}
inout	string	ContactPerson;	{if empty, will get the contact person from either the RM_Customer_MSTR table or, if CustomerAddressID is supplied, then from the RM_Customer_MSTR_ADDR table.}
inout	string	CustomerAddressID;	{if empty, then the Address Code from the RM_Customer_MSTR table}
inout	integer	NotePriority;	{if empty, then it will set it to 2}
inout	string	UserDefined1;	{if empty, and ActionPromised, will set it to UserDefined1 from ActionPromised}
inout	string	UserDefined2;	{if empty, and ActionPromised, will set it to UserDefined2 from ActionPromised}
inout	date	UserDefinedDate;	
inout	long	NextNoteIndex;	{the note index of the new note}

How to add your own letter function.

You can add support for your own letter tags. You will need to manually enter them in the body of the letter though. Say you want to add a tag called %@SalesPersonID@%, that would get the name of the salesperson for the customer. In Collection Letter Maintenance, you would need to enter something like this in the body of the letter:

Please contact %@SalesPerson%@ for information on your account.

You could use any of the other functions, etc. in the letter as well.

Create a trigger on the CN_Custom_Function procedure, on the CN_Letter_Hidden form:

```
if Trigger_RegisterProcedure(script CN_Custom_Function of form CN_Letter_Hidden,
TRIGGER_AFTER_ORIGINAL, script My_Salesperson_Trigger) <> SY_NOERR then warning
"Trigger registration for My_Salesperson_Trigger failed.";
end if;
```

The My_Salesperson_Trigger looks like this:

```
in 'Customer Number'      IN_Customer; in
'Address Code'           IN_Address;
in 'Contact Person'      IN_Contact;
in 'Currency ID'         IN_Currency;
in      boolean          IN_Consolidate_National_Account;
in      string           s_Function;
in      integer          Ai_Parameters[5];    {any parameters}
in      boolean          IN_Word_Letter;
inout boolean            IO_Failed;
inout text               IO_ReplacementText;
inout boolean            Found;

local string             Name;
local boolean            b_result;

if s_Function = "SalesPerson" then
    set 'Customer Number' of table RM_Customer_MSTR to IN_Customer;
    get table RM_Customer_MSTR;
    if not empty('Salesperson ID' of table RM_Customer_MSTR) then
        set 'Salesperson ID' of table RM_Salesperson_MSTR to
            'Salesperson ID' of table RM_Customer_MSTR;
        get table RM_Salesperson_MSTR;
        if err()=OKAY then
            Found=true;
            set Name to 'Salespersons First Name' of table RM_Salesperson_MSTR
                +' '+'Salespersons Last Name' of table RM_Salesperson_MSTR;
            if IN_Word_Letter then
                IO_ReplacementText = Name+IO_ReplacementText;
            else
                b_result = Field_InsertStringInText('(L) Letter' of window
                    CN_Letter_Hidden of form CN_Letter_Hidden, Name);
                IO_Failed = not b_result;
```

```

        end if;
    else
        Found=false;
    end if;
end if;
end if;

```

One thing of note is that for non-Word letters, the way Dexterity inserts strings into text fields requires us to process the letter from the end to the beginning, and not from the beginning to the end. For a single value replacement, like Salesperson Name, that's not important, but if you are doing a block of information like document number, document date, original amount, and remaining amount, you have to parse the fields out right to left. To get this block of text:

Document Number	Date	Original Amount	Current Amount
-----------------	------	-----------------	----------------

Your code would look something like this:

```

if IN_Word_Letter then
    IO_ReplacementText = str('Current Trx Amount' of table RM_OPEN) + IO_ReplacementText;

else

    b_result = Field_InsertStringInText('(L) Letter' of window
        CN_Letter_Hidden of form CN_Letter_Hidden, str('Current Trx Amount' of table
            RM_OPEN));
    IO_Failed = not b_result;
end if;

if IN_Word_Letter then
    IO_ReplacementText = str('Document Amount' of table RM_OPEN) +
        IO_ReplacementText;
else

    b_result = Field_InsertStringInText('(L) Letter' of window
        CN_Letter_Hidden of form CN_Letter_Hidden, str('Document Amount' of
            table RM_OPEN));
    IO_Failed = not b_result;end if;

if IN_Word_Letter then
    IO_ReplacementText = str('Document Date' of table RM_OPEN) +
        IO_ReplacementText;
else

    b_result = Field_InsertStringInText('(L) Letter' of window
        CN_Letter_Hidden of form CN_Letter_Hidden, str('Document Date' of
            table RM_OPEN));
    IO_Failed = not b_result;
end if;

```

```
if IN_Word_Letter then
    IO_ReplacementText = str('Document Number' of table RM_OPEN) +
        IO_ReplacementText;
else
    b_result = Field_InsertStringInText('(L) Letter' of window
        CN_Letter_Hidden of form CN_Letter_Hidden, str('Document Number' of
        table RM_OPEN));
    IO_Failed = not b_result;
end if;
```

In the above example, you would want to format the currency fields and dates better, but you get the idea.

Collections Management main table structure and relationships.

CN00100 – CN_Notes

This table hold all the note information, except the text of the note. It is the current version of the note. A couple of fields to note are the NoteStatus and Action Completed. These two fields sort of work together. NoteStatus is a drop down (1=Open, 2=Completed, 3=Cancelled, and 4=Cancelled by Plan). Action Completed is a boolean. When NoteStatus=1, Action Completed is false. When NoteStatus is 2, then Action Completed is true. If NoteStatus is 3 or 4, Action Complete can be either true or false, depending on whatever it was before the note was canceled. Most likely, Action Complete will be false if NoteStatus is >2. The actual text of the note is stored in the CN00300 table, linked by the note index.

CN00200 – CN_Notes_Promised

This table contains all the actions, and their settings. A couple fields are no longer used: Next Action Promised, Next Action Promised Days, Action Cancelled Date, and Action Cancelled By.

CN00300 – CN_Notes_Text

This table contains the actual text of the note. It is linked to the CN00100 by the NoteIndex field.

CN00500 – CN_Customer_Info

This table contains Collection information about the customer .

CN00600 – CN_Query_MSTR

This table contains all the query criteria for a query.

CN00700 – CN_Letters

This table contains information about Collection letters. Letter Type=3 indicates that the letter is a text letter. The CN_Word_Letter column further defines that the text letter is a Word letter.

CN00800 – CN_Plans

This table contains information about Collection Plans. Some fields are not currently used: PutOnHoldStep, and CN_PlanNoteInvoicesOption.

CN01400 – CN_Collector_MSTR

This table contains information about Collectors. The unusual aspect of this table are the two User columns; Collector and User ID. Collector ID is an all upper case field. User ID is the standard GP User ID, which is upper and lower case. Part of this is a carryover from an old product (eCollect), and also carryover from when GP only supported upper case User ID's.

Usually, these two fields are the same, except if the User ID field has lower case letters in it. We use the User ID field to find the Collector ID.

CN100100 – CN_Query

This table is used to store the Query results. Only one set of results are allowed per user at a time. There is at least one row per customer, possibly one row per customer per address id if the letter they are printing has been setup to be letter per bill to address id. The table does not store the name of the query that generated the results.

CN100200 – CN_Build_Letters

If printing text letters either from the Collection Print window, or the Query Letters window, this table gets populated with the actual text of the letter. It does not get populated if the letter is a Word letter.

CN20100 – CN_Invoice_Notes_OPEN

When an invoice is attached to a note, a row is added to this table. It contains information about the note that it is attached to, and also, about the document itself, such as the current amount, and the aging bucket it is in. These values are set at the time the invoices is attached to the note, and not updated. The current transaction amount is only used by the Mark Paid process. This amount, less the action amount, is compared to the documents actual current transaction amount in the RM Open table. If the result is less than the actual current transaction amount, the document is marked paid.

CN30100 – CN_Notes_HIST

If Save Note Revision History is marked on the Collection Setup window, as a note is revised, the previous version is copied to this table. When a note is created, a row is also created in this table, with a revision number of zero. This only tracks changes to the settings of the note (amounts, who it is assigned to, etc. It does not track changes to the contents of the note text.

CN30200 – CN_Invoice_Notes_HIST

When an invoice is marked as paid, it is moved from the CN20100 table to this table.

CN40100 – CN_SETUP

This table contains the setup information.

CN40500 – CN_Security_SETUP

This table contain information about Collectors, and what they can and cannot see, what happens when they log into GP, etc.

The Mark Paid Process

For each note that has a status of open and has an Action Type of promise.

For each note, go through the CN20100 table.

Get the actual RM Open document.

The document can be marked paid if:

- If the document isn't in the open table anymore (moved to history because it has been paid).
- If the Current Transaction Amount in the open table is zero.
- If the Current Transaction Amount in the open table is $\leq$ to the Current Transaction Amount – the Action Amount in the CN20100 table.

If the document is marked paid:

- If Save Note Revision History is not marked in Collection Setup, the note is marked as complete.
- If the Save Note Revision History is marked in Collection Setup, the note is moved to the CN30100 table, and rem