

Integrating to CDA

Core concept is that you don't have to write any Dexterity code to do this. Your 'install' routine consists of two things: Updating some SQL tables, and creating stored procedures. To distribute your CDA Add-In, the end customer will need to obtain a **CDA –Plus Version** registration key from Professional Advantage. That key will then tell CDA to look for integrating products and run your code. The customer only needs to purchase the **CDA – Plus Version** key once, regardless of how many Add-In products for CDA they have. How you update the SQL tables, or install the stored procedures is entirely up to you. What you charge for your CDA Add-In is up to you as well, and your customers will get that code from you.

The stored procedures used for archiving get installed to the GP System database, which may or may not be called DYNAMICS. Your process should also grant the necessary permissions to the stored procedure. In CDA, only SA, DYNSA, or members of the PowerUser group can perform the actual archiving. If you are supporting the CDA Data Sizing window, you will have some stored procedures at the company database as well. You need to set permissions to those stored procedures appropriately as well.

Archiving.

Add your product to the main window.IS

The API allows you to add archiving support for a product that has its own historical data (more of a vertical product) separate from GP history tables, or companion historical data (more like a horizontal product) where you are storing additional historical transactional data that link to GP history tables. Your product may have both, or only one.

Insert a row into the paISVProducts table, located in the System database.

Your installation process will need to add a row to this table for each product you want to appear. If you are only adding companion tables, you do not need to add rows to this table. The table has these columns:

ProductID – your GP product ID.

ModuleID – an integer used if you have more than one module for your product.

Display Name – this is the name that will display on windows within CDA.

Procedure Name – this is the name of the stored procedure that will perform the archive.

Your installation code will do something like the following:

```
insert into paISVProducts
(PRODID, DSPLNAME, PRCDNAME, TransferCB, RemoveCB, MODULEID) values (1755, 'Project
Accounting', 'pa_ArchiveProject', 0, 0, 0)
```

Insert rows in the paISVDBSizing table.

This table is used not only for archiving, but also by the Database Sizing window. It tells these processes which tables make up transaction history for your product. Your installation code will do something like the following:

```
insert into paISVDBSizing (PRODID, MODULEID, fileOSName, FILENAME) values
(1755, 0, 'PA30100', 'PA_Timesheet_HDR_HIST')
```

Values are: your product ID, module ID, SQL Table name, and internal table name. Insert rows for each table that you will be archiving.

Structure of your stored procedure

Your installation process will need to create this stored procedure in the system database. The name of the stored procedure must match the Procedure Name you inserted into the paISVProducts table. The stored procedure will populate two tables in the company database specified by the @ToCompany parameter. The stored procedure must have this parameter block:

Parameters:

@I_FailedTableName	varchar (80)	= NULL,
@I_RemoveTableName	varchar (80)	= NULL,
@I_Transfer	tinyint	= NULL,
@I_Remove	tinyint	= NULL,
@I_CutOffDate	datetime	= NULL ,
@I_FromCompany	varchar (40)	= NULL,
@I_ToCompany	varchar (5)	= NULL,
@I_WhichDate	smallint	= NULL,
@O_Success	tinyint	= NULL output,
@O_ErrorState	int	= NULL output

@I_Transfer is true if they are doing a transfer.

@I_Remove is true if they are doing a remove.

@I_CutOffDate is the actual date entered.

@I_FromCompany is the name of the database they are transferring data from (the live company).

@I_ToCompany is the name of the database they are transferring data to (the archive company).

@I_WhichDate – 0 if they are archiving by Document Date, 1 if they are archiving by GL Posting Date.

@O_Success – 0 if your transaction had to be rolled back, or 1 if it completed.

@O_ErrorState – if your procedure failed, return the SQL error code.

@I_FailedTableName – the name of the table you put information about rows that failed to archive. This will be [<database>]. paISVFailed.

@I_RemoveTableName – the name of the table you put information about rows that have been archived. This will be [<database>]. paISVReporting.

You may need to create temporary tables in your stored procedure with the appropriate fields/keys for the rows to be removed and rows that failed since the paISVReporting and paISVFailed tables are generic. After populating your temp tables, you can format the string fields of your table so they appear properly in the paISVFailed and paISVReporting tables accordingly.

General layout of the rest of the stored procedure:

1. Start a transaction.
2. Create and populate the ##RemoveTable based on the main table for your transactional data. Create another table with the same definition and call it ##TransferTable. You define what these temp tables look like in your procedure. Typically you are reading from a header table like the SOP30200. What rows you put in there is based on the type of cutoff date passed in, either Document Date or Posting date. The @I_WhichDate parameter will be 0 for Document Date, 1 for Posting Date. Your product may not have a Posting date, so you may not need to check this parameter.
3. Now that you have your list of header keys, do any criteria checking, removing those rows that do not pass. The rows that do not pass can be inserted into the paISVFailed table, along with a reason.

The paISVFailed table has four columns, ProdID (your GP Product ID), STRNG132 (a string that you can put in whatever represents the key fields for the document), and ERRDESCR (a string describing why it wasn't archived), and Module ID (an integer for the module of your product, 0 if you don't have modules).

4. Copy rows from the ##RemoveTable that do not exist in the archive company to the ##TransferTable. The transfer table now contains documents that are not in the archive company, and the remove table now contains rows to be removed from the live company. Copy the rows from the ##RemoveTable to the paISVReporting table, which has your ProductID, STRNG132 (a string that identifies this document, like the document number plus the type, plus the date for instance), and STRGA255 for a description of the document, and Module ID (the module of your product being archived, zero if you don't have modules.)

A diagram of the process looks kind of like this:



5. Count rows in the source and destination tables before we start:


```

exec pa_AddSQLTableCounts 'TableXXX', @I_ToCompany, 1
exec pa_AddSQLTableCounts 'TableXXX', @I_FromCompany, 3
      
```

 Replace TableXXX with the name of your SQL table.
6. The next block is the actual archiving. There are two parameters @Transfer and @Remove. Customer may be doing a transfer only, a remove only, or a transfer and remove, so we have something like this:


```

If @Transfer
    Insert into @I_ToCompany.TableXXX select from @I_FromCompany.TableXXX Src where
    Src.Key = TransferTable.Key
If @Remove
    Delete @I_FromCompany.TableXXX from RemoveTable where
    @I_FromCompany.TableXXX.Key = RemoveTable.Key
      
```
7. Recount the data in the live and archive databases


```

exec pa_AddSQLTableCounts 'TableXXX', @I_ToCompany, 2
      
```

exec pa_AddSQLTableCounts 'TableXXX', @I_FromCompany, 4
Replace TableXXX with the name of your table.

8. If doing a transfer only (if @I_Transfer and not @I_Remove), copy ##TransferTable to the palSVReporting table, otherwise, copy ##RemoveTable to the palSVReport table.
9. Check to see if any errors happened, and if so, roll the transaction back.

When archiving is completed, CDA will print a report based on the ISVReporting table, and another report based on the palSVFailed table, similar to what it prints for other module archived.

If you don't need to apply additional criteria to archive your companion data, you only need to add one row, and create one stored procedure. The stored procedure will be in the system database (DYNAMICS, or some other name). You will need to give DYNGRP execute permissions to it.

Your stored procedure will have these parameters:

@RemoveTableName	varchar(40)	= NULL,
@FailedTableName	varchar(40)	= NULL,
@Transfer	tinyint	= NULL,
@Remove	tinyint	= NULL,
@CutOffDate	datetime	= NULL ,
@FromCompany	varchar (40)	= NULL,
@ToCompany	varchar (5)	= NULL,
@WhichDate	smallint	= NULL,
@Success	tinyint	= NULL output,
@ErrorState	int	= NULL output

@RemoveTableName is the SQL table name that contains the list of documents to be removed. If you are filtering, the rows you remove from this table, get inserted into the @FailedTableName table.

@FailedTableName is the name of the table that contains documents that were within the cutoff date range, but could not be archived because of your criteria.

@Transfer is true if they are doing a transfer.

@Remove is true if they are doing a remove.

@CutOffDate is the actual date entered.

@FromCompany is the name of the database they are transferring data from (the live company).

@ToCompany is the name of the database they are transferring data to (the archive company).

@WhichDate – 0 if they are archiving by Document Date, 1 if they are archiving by GL Posting Date.

@Success – 0 if your transaction had to be rolled back, or 1 if it completed.

@ErrorState – if you procedure failed, return the SQL error code.

The table definition for @RemoveTableName and @FailedTableName varies by module. See Appendix B for details.

If you are only filtering the list of documents, your procedure will be pretty simple – move rows from the RemoveTable, and insert them into the FailedTable. The failed table has a column for putting in a description of why they won't archive.

Adding your tables to the GP module archive process.

This is for the case where your product has tables that are related to the GP modules transaction history tables. For instance, if you have some additional SOP document information, you want that additional information to be available in the archive company, as well as removed from the live company. In CDA, some modules have more stringent requirements than just the date before they can be archived. RM documents need to be fully applied, and all the documents in the apply chain must also be fully applied and within the cutoff date. If your product needs to enforce some criteria, the API provides for this. In general, CDA archive process is to populate a temp table with the keys of rows to archive. Before starting the actual movement of data from the live company, CDA can call your procedure, passing in the table with the keys. Your procedure can then remove any rows that don't meet your criteria. CDA then looks at the remaining keys in the table, and archives those documents.

Your installation code would write values to the paISVModComp table that looks like this:

- Product ID – your GP Product ID.
- Module – a string representing which module archive you want to trigger on. See Appendix C for the list of strings and what they map to.
- TypeID – an integer. 1 if you want to filter the list of documents further, 2 for archiving your tables.
- Procedure – the stored procedure to call.

Example:

```
insert into paISVModComp values
(1755, 'RM', 1, 'paProjectModuleCompanionFilter') /* filter further */
insert into paISVModComp values
(1755, 'RM', 2, 'paProjectModuleCompanionArchive') /* archive additional tables
*/
```

In this example, we are both adding criteria for documents to be archived, as well as archiving additional tables in the companion product. Both stored procedures needs this parameter block:

```
@RemoveTableName    varchar(40) = NULL,
@FailedTableName    varchar(40) = NULL,
@TempTable1Name     varchar(40) = NULL,
@TempTable2Name     varchar(40) = NULL,
@TempTable3Name     varchar(40) = NULL,
@Transfer           tinyint      = NULL,
@Remove            tinyint      = NULL,
@CutOffDate        datetime     = NULL ,
@FromCompany       varchar (40)  = NULL,
@ToCompany         varchar (5)   = NULL,
@WhichDate         smallint     = NULL
```

The remove table is the name of the table that contains rows to be archived.

The FailedTableName is the name of the table that contains rows that were not archived, and why not.

The three TempTableName strings are additional temp tables. Some modules (like Bank Rec), populate multiple temp tables in order to determine what rows to archive.

Transfer and Remove are the flags from the window, as well as the CutOffDate.

From and To Company are the names of the databases respectively.

WhichDate is the date option 0=DocDate, 1= GL Posting Date selected on the archive window.

The filter procedures intent is to look at the contents of the RemoveTableName table, and remove rows from it, inserting them into the FailedTableName table, with a description.

Example:

```
select @ExecString = 'insert into '+@FailedTableName+' select R.RMDTYPAL,
R.DOCNUMBR,R.CUSTNMBR,R.DOCDATE, ''Starts with letter A'','+
''','', '','', ''+ '01/01/1900''+ ' from '+@RemoveTableName+' R where
CUSTNMBR in (select CUSTNMBR from '+@FromCompany+'..RM00101 where CUSTNAME
like ''A%'')'
exec (@ExecString)
select @ExecString = 'delete '+@RemoveTableName+' where CUSTNMBR in (select
CUSTNMBR from '+@FromCompany+'..RM00101 where CUSTNAME like ''A%'')
```

If you are archiving data in your own tables, you need to add those tables to the paISVDBSizing table. In the example above, we have it being a companion to the RM module archive, so the SQL would look like this:

```
insert into paISVDBSizing (PROID,MODULEID, fileOSName,FILENAME) values
(1755, 5, 'PA30100', 'PA_Timesheet_HDR_HIST')
```

Values are: your product ID, module ID, SQL Table name, and internal table name. Insert rows for each table that you will be archiving.

Adding your tables to the archive master data process.

Master data, is data that is either setup, or base entity related. A base entity would be Customer, Vendor, Item, etc. This includes things like tax schedule and details, report options, module setup, configuration tables, etc. Master data archiving is transfer only. It is never removed from the live database. Existing master data in the archive database is updated, and new rows from the live database are added to the archive company. CDA will build and execute SQL to properly update and transfer new for you. All you need to do is tell it the table name. It will use the primary key on the table. To do this, your installation needs to put a row in the paISVMasterData table:

```
insert into paISVMasterData (PROID,fileOSName,fileName) values (1755,
'PA00401', 'PA_Fee_MSTR')
```

The values are your product ID, the SQL table name, and the internal name of the table. Repeat this code for each table you want to add.

Adding criteria to the Will it Archive logic.

This window emulates checking the apply chain for RM and PM documents for a single document. Customers get the report at the end of the archive that says these documents failed to archive, but finding the one document in an apply chain of thousands of documents can be incredibly tedious using the GP inquiry windows. The window displays all the documents, and flags which ones are holding up the archive. You may have additional requirements for a document to be archived, but the window is only looking at the apply chain. At this point, integrating to the window is not supported. If this is something of interest, please contact Professional Advantage support.

Preview Window

The preview window off of the main archive window gives rough numbers of rows. Getting more accurate numbers is the job of the CDA Data Sizing window. The concept is to provide some quick feedback based on a date of the volume of data that would be archived. Customers may then chose an earlier or later date depending on the results. With that, it is a count of the rows in the header table only for that module. Integrating to this window really only applies if you are adding your product to CDA. To integrate to this window, your installation code would add a row to the paISVPreview table:

Product ID – your GP product ID.

Display Name – the name of your product.

Procedure – name of your stored procedure.

Module ID – integer of the module within your product, zero if you don't have modules.

Your procedure will need the following parameters and look something like this:

```
CREATE procedure [dbo].[pa_PreviewProject]
    @I_Company          varchar(40)          = NULL ,
    @I_CutOffDate        datetime            = NULL ,
    @I_WhichDate         int                 = NULL
as
declare @ExecString varchar (2000),
        @t_Rows int

select @t_Rows = 0
create table #RowCountTable
(
    NumberOfRows         int
)
/*
    Project only has the one date, so we don't care
    if @I_WhichDate=0
    else
*/
select @ExecString = 'insert into #RowCountTable select count(*) from
'+@I_Company+'..PA30100 C where C.PADOCDT <= '''+
rtrim(convert(char(12),@I_CutOffDate,112)) + ''''
exec (@ExecString)
select @t_Rows = (select distinct NumberOfRows from #RowCountTable)
delete #RowCountTable

select @ExecString = 'insert into #RowCountTable select count(*) from
'+@I_Company+'..PA30200 C where C.PADOCDT <= '''+
rtrim(convert(char(12),@I_CutOffDate,112)) + ''''
exec (@ExecString)
select @t_Rows = @t_Rows+(select distinct NumberOfRows from #RowCountTable)
delete #RowCountTable

/* Now, insert total into paISVPreview table */

select @ExecString = 'update DYNAMICS..paISVPreview set Count1 =
'+convert(char,@t_Rows)+' where PRODID=1755 and MODULEID=0'

exec (@ExecString)
```

CDA Data Sizing window

This window is intended for a more accurate count of data. It's not exact, but it's closer than the preview window. The concept is that you not only count rows in the header table, but in all the related tables as well for the documents in the date range. So we not only count SOP30200 rows, but also rows in the SOP30300, SOP30201, etc. You can have your product appear on the window, and you can also have the number of rows in your companion tables added to counts for the standard GP modules. The window passes in a succession of cut off dates, a year apart, ending with the current year.

Your installation code will also need to add a row to the paISVDBSizingLogic table:

ProductID – your product ID.

ModuleID – Module ID, either for GP if you are doing a companion module archive, or your module id if you have modules in your product.

Procedure Name

Example:

```
insert into paISVDBSizingLogic (PRODID,MODULEID, PRCDNAME) values (1755,0,'pa_DBsizingProject')
```

You can add multiple rows if you have companion data in multiple modules, or your product not only has its own data, but also companion tables as well. Your procedure will calculate row counts for each table, for each year, and update the paDBDist table.

Your procedure needs to be created in the DYNAMICS or system database, and will need the following parameters:

```
@Company    varchar(40),
@Year       int,
@StartDate  datetime,
@EndDate    datetime
```

The structure of the procedure depends on if you are doing your product, or doing a companion data. If you are doing your product, the steps are typically:

1. Create a temp table with key info for the header table.
2. Insert into it key information from the header table.
3. Count the rows in the temp table, and insert a row in the paDBDist table for the header table.
4. Count the number of rows in each related table that have matching keys in the temp table, and add a row to the paDBDist table for that table.
5. Repeat step 4 for each table that is part of your module.

You may need to create and populate additional temp tables if your product has multiple header type tables.

Since your proc resides in a database different than the tables, the @Company parameter is the name of the database where the tables are found. With that, you have to build a string, and then execute that string. To insert a row into the paDBDist table:

```
• declare @ExecString varchar(2000)
• /* Step 1 */
• create table #ProjectTemp
• (
•     ProjectNo    CHAR(17)
• )
• create table #RowCountTable
• (
•     NumberOfRows    int
• )
• select @ExecString = 'insert into #ProjectTemp select PATSNO from '+@I_Company+'..PA30100 S where S.PADOCDT between '''+
    rtrim(convert(char(12),@I_StartDate,112)) + '''+
```


- `' and '''+ rtrim(convert(char(12),@I_EndDate,112)) + '''`
- `exec (@ExecString)`
- `select @RCount = (select count(*) from #ProjectTemp)`
-
- `select @ExecString = 'insert into '+@I_Company+'..paDBDist values ('+convert(char,@I_Year) +',1755,0,'PA30101',''+ convert(char,@Rcount)+' ,0) '`
- `exec (@ExecString)`
-
- `exec('insert into #RowCountTable select count(*) from '+@I_Company+'..PA30101 C where C.PATSNO in (select ProjectNo from #ProjectTemp) ')`
- `select @Rcount = (select distinct NumberOfRows from #RowCountTable)`
- `delete #RowCountTable`
-
- `select @ExecString = 'insert into '+@I_Company+'..paDBDist values ('+convert(char,@I_Year) +',1755,0,'PA30101',''+ convert(char,@Rcount)+' ,0) '`
- `exec (@ExecString)`

The columns in the paDBDist table are Year, product ID, Module ID, table name, and the number of rows.

If you are doing companion tables, your SQL will need to do something like this:

`@RowCount = Select count(*) from MyTable where docnumber in (select docnumber from GPHeaderTable where GPHeaderTable.docdate between @StartDate and @EndDate)`

Appendix A – GP Module ID

- 1 – General Ledger
- 2 – Bank Reconciliation
- 3 – Payables Management
- 4 – Purchase Order Processing
- 5 – Receivables Management
- 6 – Invoicing
- 7 – Sales Order Processing
- 8 – Inventory Control
- 9 – Payroll
- 10 – Bill Of Materials

Appendix B – Module temporary file names and contents

GL – none.

BR:

RemoveTable:	Random named Dex temp table.
	CHEKBKID
	RECONUM
	CMRECNUM
	CMTrxNum
FailedTable:	pacmF
Temp1:	pacmt001 Headers from the CM20500 table.

PM :

RemoveTable:	papmt001
FailedTable:	papmtF
Temp1:	Random named by Dex temp table with same structure as papmt001. Contains transactions to be archived.

PM Batches:

Transactions:	papmt003 Batches
---------------	-----------------------

POP POs:

RemoveTable:	papopt01
FailedTable:	papopF
Temp1:	Random named by Dex temp table with same structure as papopt01. Contains transactions to be archived.

POP Receipts

RemoveTable:	papopt02
FailedTable:	<none>
Temp1:	Random named by Dex temp table with same structure as papopt02. Contains transactions to be archived.

RM:

RemoveTable:	parmt001
FailedTable:	parmF
Temp1:	Random named by Dex temp table with same structure as parmt001. Contains transactions to be archived.

RM Batches:

RemoveTable: parmt003
FailedTable: parmF
Temp1: Random named by Dex temp table with same structure as parmt003.
Contains transactions to be archived.

IVC :

RemoveTable: paivct01
FailedTable: paivcF
Temp1: Random named by Dex temp table with same structure as paivct01.
Contains transactions to be archived.

SOP :

RemoveTable: pasopt01
FailedTable: pasopF
Temp1: Random named by Dex temp table – same structure as pasopt01.
Contains transactions to be archived.

IV:

RemoveTable: paivt01
FailedTable: paivF
Temp1: Random named by Dex temp table with same structure as paivt01.
Contains transactions to be archived.

UPR:

RemoveTable: pauprt01
FailedTable: <none>
Temp1: pauprt02
Contains transactions to be archived.

BOM:

Temp1: pabmt001

Appendix C – Module Companion Archiving codes

GL	-	General Ledger
BR	-	Bank Reconciliation
RM	-	RM Transactions
RMB	-	RM Batches
PM	-	PM Transactions
PMB	-	PM Batches
SOP	-	SOP Transactions
BOM	-	Bill Of Materials
IVC	-	Invoicing
IV	-	Inventory
POP	-	Purchase Orders
POR	-	Purchase Receipts
UPR	-	Payroll