TIPS FOR A PRO DATASET DEVELOPER

Introduction

There have been a number of major milestones in Progress releases. The first that comes to mind in more recent times was the introduction of the appserver. Since then there have been other significant introductions such as super procedures, processing handles, and object orientated code. Another recent significant feature, which is the text of this paper, is the prodataset.

The dataset's main advantage is the ability to encapsulate data, both related and unrelated, and share that data between procedures, internal procedures, and environments as a single parameter. Between environments the dataset is copied from one dataset object to another, but within the one environment a dataset can be shared between procedures. A procedure can access that data dynamically, or procedurally using standard 4GL coding and as no physical data passes there is virtually no overhead.

Another significant feature of the prodataset is the captured changes log that is used to track changes and ensure a user doesn't update another's changes.

The following goals have been listed under the acronym ALIAS. Just as an alias provides another name for an item the dataset provides another view of the organisation data.

Goals

Avoid common errors and misconceptions.

Learn prodataset capabilities through operational examples of their use.

Identify principals of Prodataset design.

Answer the why Prodataset's.

Shortcut coding consideration with commercial code examples.

Capabilities by example

1. **Share your transport**

   In a framework I developed all common tables are held in 1 dataset. This dataset can be shared with any procedure within the same environment.

   The definition for the common client PDS is below. The ttwindow and ttwindowfield includes the details of every client window and field currently running on the client. The ttwindowfield include the handle of every object on the window and the handle to the buffer field that populated it.

```
{adf/adfttmenu.i &ReferenceOnly={&ReferenceOnly}}
{adf/adfttbutton.i &ReferenceOnly={&ReferenceOnly}}
{adf/adfttholiday.i &ReferenceOnly={&ReferenceOnly}}
{adf/adfttcomposite.i &ReferenceOnly={&ReferenceOnly}}
{adf/adfttcomponent.i &ReferenceOnly={&ReferenceOnly}}
{adf/adfttlist.i &ReferenceOnly={&ReferenceOnly}}
{adf/adfttsyparam.i &ReferenceOnly={&ReferenceOnly}}
{adf/adfttappserver.i &ReferenceOnly={&ReferenceOnly}}
{adf/adfttwindow.i &ReferenceOnly={&ReferenceOnly}}
{adf/adfttwindowfield.i &ReferenceOnly={&ReferenceOnly}}
{adf/adfttcontext.i &prefix=Client
&ReferenceOnly={&ReferenceOnly}}

DEF DATASET dsAdf {&ReferenceOnly} FOR
TTadfMenu,TTadfbutton,TTadfSyParam,TTadfHoliday,TTadfComposite,TT
adfComponent,TTadfList,TTWindow,TTWindowField,TTappServer,TTClien
tContext
    DATA-RELATION adfComponent FOR TTadfComposite,TTadfComponent
      RELATION-FIELDS (procName, procName).
```

   One of these tables is a context temp-table that holds variable data that can be shared between internal-procedures and procedures. The context table replaces many variables and variable are only used for temporary local control. Using a context dataset is discussed next.

2. Travel by public transport.

The advantage of passing a prodataset is that you can limit the parameters passed and can easily add parameters. Pass two datasets, one relating to the function being performed and the other a context dataset. The context dataset may include a context temp-table and a message temp-table. An example is below.

```
DEF TEMP-TABLE TT{&prefix}context NO-UNDO {&ReferenceOnly}
    FIELD contextId          as CHAR
    FIELD contextName        AS CHAR
    FIELD contextValue       AS CHAR
    INDEX contextName IS PRIMARY UNIQUE contextId contextName.
DEF TEMP-TABLE TT{&prefix}message NO-UNDO {&ReferenceOnly}
    FIELD messageType AS CHAR
    FIELD messageText AS CHAR
    INDEX messageType IS PRIMARY UNIQUE messageType.
DEF DATASET ds{&prefix}Context {&ReferenceOnly} FOR
TTcontext,TTmessage.
```

The contextId in the above example allowed the context to be associated to a particular object such as a window or procedure handle. If the context is not related to a specific object then the Id is left blank.

To simplify usage create a super setContext and getContext.

3. **Use the most appropriate transport package**

You can pass a dataset-handle and receive a dataset and visa-versa. The following is a standard run statement from a framework that highlights this.

```
RUN VALUE(bTTWindow.serverProc) ON bTTappServer.asHandle
(cAction,INPUT-OUTPUT DATASET dsContext,INPUT-OUTPUT DATASET-
HANDLE hDsHandle).
```

It is received in a procedure below as a dataset.

```
DEF INPUT        PARAMETER cAction AS CHAR NO-UNDO.
DEF INPUT-OUTPUT PARAMETER DATASET-HANDLE hContextDs.
DEF INPUT-OUTPUT PARAMETER DATASET FOR dsFeed.
```

Notice the ‘FOR’ in the last line that allows you to receive a dataset with a known definition.

The dataset can also be passed as an ordinary handle but then received as a handle or dataset-handle.

One of the advantages of passing a handle or a dataset handle is that you can pass different datasets. In the client framework example above, all datasets are passed via the dataset handle `hDsHandle`. On the server the dataset is received with a known construction. In other examples the server receives the unknown dataset handle which it then processes as a fill or save without real knowledge of the underlying data.

4. Just give me the transport link

The main advantage of the datasets is the ability to share the dataset between procedures. This is achieved by passing the PDS by reference or bind and basically means that you are passing the handle or a pointer to where the actual data is stored.

Beware that when you pass a dataset to an internal procedure it only remains in scope within that IP.

The easiest way to check if you are receiving a copy of the dataset is to check the handles. Run the following example with the BY-REFERENCE commented and used. When commented you will notice that you actually have a copy of the dataset rather than receiving and using the original dataset.

```
/* Examples/Handles1.p */
DEF VAR hBuffer AS HANDLE NO-UNDO.

DEF TEMP-TABLE ttCustomer NO-UNDO
    FIELD custnum AS INTEGER
    FIELD NAME AS CHARACTER
    INDEX bynum IS UNIQUE custnum.
DEF DATASET dsCustomer FOR ttCustomer.

MESSAGE 'Before' DATASET dsCustomer:HANDLE VIEW-AS ALERT-BOX.
RUN passTT (DATASET dsCustomer /*BY-REFERENCE*/ ).

PROCEDURE passTT:
    DEF INPUT PARAMETER DATASET-HANDLE hDsCustomer.
    MESSAGE 'After' hDsCustomer VIEW-AS ALERT-BOX.
END.
```

5. **It gets you there fast**

While I have not noticed much difference on fill times compared to a manual fill, passing datasets by-reference certainly removes the overhead in copying data from one procedure to another.

But, probably the biggest time saver is in reduced coding. See the following fill example.

```
DEF TEMP-TABLE TTcustomer NO-UNDO
    FIELD customerNo  AS INT
    INDEX customerNo IS PRIMARY UNIQUE customerNo.
DEF DATASET dsAdf FOR TTcustomer.

DEF DATA-SOURCE srccustomer FOR customer.
BUFFER ttcustomer:ATTACH-DATA-SOURCE(DATA-SOURCE
srccustomer:HANDLE).
DATASET dsadf:FILL().
BUFFER ttcustomer:DETACH-DATA-SOURCE().
```

6. But I like shanks pony

Although you may pass a dataset handle using the FOR verb in parameter statements (e.g. DEF INPUT-OUTPUT PARAMETER DATASET FOR dsCustomer) allows the developer to use standard (non dynamic) code by referring to the TT's defined in the dataset. The TT's and dataset should be defined as REFERENCE-ONLY as described earlier (but this is not a mandatory requirement). The PDS definition is used for compiling only and at runtime the received or output created dataset is used. The definition and the received dataset must have the same number of tables and fields, and the field types must be correct and in the same order. The names however may vary.

In the following example the dataset is passed a handle operated on and passed again by-reference and displayed using standard 4GL coding.

```
/* Examples/Handles2.p */
DEF VAR hBuffer AS HANDLE NO-UNDO.

DEF TEMP-TABLE ttCustomer NO-UNDO
    FIELD custnum  AS INTEGER
    FIELD custname AS CHARACTER
    INDEX bynum IS UNIQUE custnum.
DEF DATASET dsCustomer FOR ttCustomer.

RUN passTT (DATASET dsCustomer BY-REFERENCE).

PROCEDURE passTT:
    DEF INPUT PARAMETER DATASET-HANDLE hDsCustomer.

    DEF VAR hCust AS HANDLE NO-UNDO.

    DO TRANSACTION:
        hCust = hDsCustomer:GET-BUFFER-HANDLE(1).
        hCust:BUFFER-CREATE.
        ASSIGN
            hCust:BUFFER-FIELD('custnum'):BUFFER-VALUE = 1
            hCust::custname = 'Test'.
    END.
    RUN passTT2 (DATASET-HANDLE hDsCustomer BY-REFERENCE).
END.

PROCEDURE passTT2:
    DEF INPUT PARAMETER DATASET FOR dsCustomer.

    MESSAGE ttCustomer.custnum ttCustomer.custname VIEW-AS ALERT-
BOX.
END.
```

Some Principles of Prodataset Design

1. Define Temp Tables (TT's) as an include and do not use LIKE

The advantage of using LIKE to define a TT within a dataset is that database changes are automatically inherited by the PDS. However –

(1) The effect of any change may not be apparent to the developer and the client may receive information they shouldn't.
(2) It also means that the database must be connected when developing client procedures. (While I might usually have a dataset connection I can remove the connection to confirm that I haven't mistaken referenced a dataset record).
(3) Creating the TT independent of the database also makes you think how you might package the fields for the dataset i.e. the dataset should be designed for the client/application instance and not necessarily reflect just how the data is stored in the database.
(4) A client procedure may also connect to a number of different databases and sources and each source may use different tables and/or fields to populate the data. The client may define tables such as users or staff, system parameters, products, customers, debtors, or vendors etc and each of these may be populated from the different sources using for example pair list.
(5) Finally by using an include, the TT can be used independent of the PDS.

Example

```
DEF TEMP-TABLE TTadfHoliday{&ext} NO-UNDO {&ReferenceOnly}
    BEFORE-TABLE biTTadfHoliday
    FIELD holiday            AS DATE
    FIELD holidayDesc        AS CHAR
    FIELD state              AS CHAR
    {adf/adfttrow.i}
    INDEX holiday IS PRIMARY UNIQUE holiday.
```

When creating a temp-table name I suggest that it is created in a standard way such as TT followed by the database table name.

2. Allow for name variations

When using PDS's you quickly find that you end up with a couple of datasets in the one procedure that have common TT's. Naming your TT's appropriate to client needs reduces this problem. It is also good practice to add an include file reference, as shown in the previous example (i.e `{&ext})`, to use where needed. Ensure you also add the extension to the PDS definition as well e.g. DEF DATASET dsHoliday{&ext} `{&ReferenceOnly}` FOR `TTadfHoliday{&ext}`.

I also recommend that you name your extension is such a way that you can easily identify the extension. I begin the extension with an underscore and this allows me to match a TT name to the base name by entry(1,<TTname>,'_').

The data source is mapped to the dataset by name but between procedures datasets are mapped by position, sequence, datatype, and key name. This allow for clearer field naming particularly at the client level. For example the following dataset dsTTsyparam maps quite happily to the TTworkgroup dataset, even if is passed by reference.

```
DEF TEMP-TABLE TTworkgroup NO-UNDO
   BEFORE-TABLE biTTworkgroup
   FIELD groupID      AS CHAR
   FIELD groupName  AS CHAR
   INDEX groupID IS PRIMARY UNIQUE groupID.
DEF DATASET dsWorkGroup FOR TTworkgroup.

RUN procedure.p (OUTPUT DATASET dsWorkGroup BY-REFERENCE).

/* procedure.p */

DEF TEMP-TABLE TTsyparam NO-UNDO
   BEFORE-TABLE biTTsyparam
   FIELD paramID     AS CHAR
   FIELD charValue   AS CHAR
   INDEX paramID IS PRIMARY UNIQUE paramID.
DEF DATASET dsTTsyparam FOR TTsyparam.

DEF OUTPUT PARAMETER DATASET FOR dsTTsysparam.
```

3. Include a {&referenceOnly} argument in each TT and PDS definition

A reference only dataset is inserted into a procedure to allow the developer to compile using (non dynamic) standard 4GL code. At run-time the reference only dataset is not used as the data is 'received' for the source procedure. Include a {&referenceOnly} argument in every TT and dataset definition so that it can be used in a reference-only context.

Reference-only is not a mandatory attribute in all circumstances but setting it provides documentation as to the intent and prevents misuse of the dataset i.e. if the dataset is flagged as reference-only then it can't accidentally be used as the dataset.

The following example uses a TT only, but the same principle applies to datasets. Test as is. Uncomment REFERENCE-ONLY and retest. Note how the message in the main block fails. So using REFERENCE-ONLY ensures that if you do use dataset outside the scope of the IP it will fail (be that the main block or another IP without a FOR ttCustomer). Take particular notice of the handles.

```
/* examples/reference1.p */
DEF VAR hCalled AS HANDLE NO-UNDO.

DEF TEMP-TABLE ttCustomer NO-UNDO
    FIELD custnum  AS INTEGER
    FIELD custname AS CHARACTER
    INDEX bynum IS UNIQUE custnum.

RUN examples/reference2.p PERSISTENT SET hCalled.
SESSION:ADD-SUPER-PROCEDURE(hCalled).
MESSAGE 'TT sent' BUFFER ttCustomer:HANDLE
     VIEW-AS ALERT-BOX.
RUN passTT (INPUT TABLE ttCustomer BY-REFERENCE).

/* examples/reference2.p */
DEFINE TEMP-TABLE ttCustomer NO-UNDO REFERENCE-ONLY
    FIELD custnum  AS INTEGER
    FIELD custname AS CHARACTER
    INDEX bynum IS UNIQUE custnum.

DEF VAR hBuffer as handle NO-UNDO.
hBuffer = BUFFER ttCustomer:HANDLE no-error.
MESSAGE 'Valid REFERENCE-ONLY' VALID-HANDLE(hBuffer)
     VIEW-AS ALERT-BOX.

PROCEDURE passTT:
    DEFINE INPUT PARAMETER TABLE FOR ttCustomer.

    MESSAGE 'TT used' BUFFER ttCustomer:HANDLE
             VIEW-AS ALERT-BOX.
END.
```

4. Consider the indices you need on the TT's

Try to always define your primary unique index for each table. If you use a fill-mode of merge or replace OR a copyDatset replace-mode note the comment from the documentation - "based on the table's unique primary index".

Keep you indices to a minimum. Usually the client has a small number of records and the primary index may be sufficient enough to enforce uniqueness and provide the right sort.

Also consider if a key should be unique. At a client a decision was made that only required data was sent to a web client. In one instance this meant that part of a primary key was not populated, and to facilitate this, the primary key had to be non-unique. Also, if an index is defined as unique at the client then your client framework must allow for this otherwise you may get uncontrolled Progress errors you do not want.

5. Add a standard include to each TT including database rowid

Add a standard include to each TT as shown in an earlier example.

```
/* adf/adfttrow.i */
FIELD dbRowid         AS CHAR
FIELD lastTime        AS DATETIME
INDEX dbRowid dbRowid
```

The first field dbRowid holds a rowid list of rowids of the data-source rows that built the TT. The rowid's are populated at each row fill and save (as part of a framework) and are used primarily to reposition within the database query to refresh the data.

Rowid's may be assigned to each TT by an undocumented feature of the pairs list of ATTACH-DATA-SOURCE. The following is an example.

BUFFER TTorder:ATTACH-DATA-SOURCE(<DataSourceHandle>, "orderRowid, rowid(order), customerRowid, rowid(customer)").

lastTime stores the time the row was retrieved and is used to determine whether the row should be refreshed which is a configurable attribute of each window that hold the dataset.

A dbRowid index has also been added to each TT to allow a fast retrieval of a row when the dbRowid is known. If the local data is refreshed (by and empty and fill) the local rowid's may change, but the dbRowid's are constant.

Other Prodataset coding tips

1. Consider the fill mode to use

The default fill-mode is MERGE and merge checks if an added record is available based on the TT unique primary index. Only unavailable records will be added and no errors or warnings are displayed on records already present.

This caused an issue in two example instances. The first was when choosing to provide an include-list on the data-source that excluded a field in a unique primary index. Only 1 record was returned as the unpopulated field are set with the initial value which meant all records other than the first failed to add. This was solved by removing unique on the primary key.

In the second instance the TT was populated from a join of two tables and the second table and a field was part of the primary key in the first. The field in the second table had not been used and had the initial value. On the default mapping for that field on the data-source was the second table and again I ended up with 1 record. The solution here was to set pairs-list on the data-source to the first table.

2. Stoping a fill

To stop a TT fill set the FILL-MODE of the TT buffer to ‘NO-FILL’ or set the ACTIVE attribute of the data-relation to FALSE.

When you set a TT to no-fill (particularly a relation parent) ensure that the data-relation is made not active, otherwise the TT will not fill. You may choose to effect this in your framework e.g.

```
DO iRelation = 1 TO phDataSet:NUM-RELATIONS:
    hRelation = phDataSet:GET-RELATION(iRelation).
    IF hRelation:ACTIVE AND
     (hRelation:CHILD-BUFFER:FILL-MODE = 'NO-FILL'
      OR hRelation:PARENT-BUFFER:FILL-MODE = 'NO-FILL') THEN
        ASSIGN
            hRelation:ACTIVE = NO
            cRelation = cRelation + ',' + STRING(hRelation).
END.
hDataSource:FILL-WHERE-STRING = cFillWhereString.
phDataSet:FILL().
DO iRelation = 2 TO NUM-ENTRIES(cRelation):
    ASSIGN
        hRelation = WIDGET-HANDLE(ENTRY(iRelation,cRelation))
        hRelation:ACTIVE = YES.
END.
```

Always ensure the relations are adjusted before any fill-where-string is applied as the adjusting of the relations seems to clear any fill-where-string.

3. Refresh a TT row

As explained earlier a TT field/column may be added to hold the source rowid for the row e.g. dbrowid. To refresh the row simply set the RESTART-ROWID to the stored source rowid and set BATCH-SIZE = 1 and have the TT or dataset FILLed.

4. Let the data-source resolve the query and use fill-where-string

Define the data-source and let the fill resolve the query.

```
DEF TEMP-TABLE TTcustomer NO-UNDO
    FIELD customerNo  AS INT
    INDEX customerNo IS PRIMARY UNIQUE customerNo.
DEF DATASET dsAdf FOR TTcustomer.

DEF DATA-SOURCE srcCust FOR customer.
BUFFER ttcustomer:ATTACH-DATA-SOURCE(DATA-SOURCE srcCust:HANDLE).
DATASET dsadf:FILL().
BUFFER ttcustomer:DETACH-DATA-SOURCE().
```

Use the data-source FILL-WHERE-STRING attribute to filter the query as required. The unspecified fill-where-string is null but to clear the filter assign the fill-where-string as below

```
DATA-SOURCE srccustomer:FILL-WHERE-STRING = 'WHERE'.
```

5. Add an inactive relation to fill up from child

To fill a dataset from a query on a child add a “NOT-ACTIVE” relation to the dataset child to parent and make the relation active when querying a child.

```
DEF DATASET dsAdf FOR TTadfComposite,TTadfComponent
    DATA-RELATION adfComponent FOR TTadfComposite,TTadfComponent
      RELATION-FIELDS (procName, procName)
    DATA-RELATION adfComposite FOR TTadfComponent,TTadfComposite
      RELATION-FIELDS (procName, procName)
      NOT-ACTIVE.
```

An example FILL-WHERE-STRING follows

```
DATASET dsadf:GET-RELATION("adfComponent"):ACTIVE = NO.
DATASET dsadf:GET-RELATION("adfComposite "):ACTIVE = YES.
DATA-SOURCE srcadfComponent:FILL-WHERE-STRING = "WHERE
componentName BEGINS 'tb'".
```

6. **Add indexed-reposition when restart-rowid is used**

   At the time writing OE10.1C performed poorly when RESTART-ROWID was used to reposition a prodataset data-source to a rowid that was deep within the default query. To avoid this problem ensure that INDEXED-REPOSITION is specified and the fill-where-string can be used to do that. The example below shows the modification to the previous customer data-source.

   ```
   DATA-SOURCE srccustomer:FILL-WHERE-STRING = 'INDEXED-REPOSITION'.
   ```

7. **Use an include field list rather than an exclude list**

   Should you wish to filter the fields that are populated with data, it is recommended that you should probably use an include field list rather than an exclude and as a newly added field will be populated unless added to the exclude list.

8. Empty a dataset when filling from a persistent procedure

Empty the dataset in procedure that returns a dataset that is defined in the persistent procedure and not passed by reference. As a TT must be defined in the main block this is not unlike having to initialize a variable defined in main and reused in an IP. As you can't be sure of the current state of the defined dataset it is best to ensure tracking-changes is false and any changes are rejected. The following code illustrates.

```
/* examples/refill1.p */
DEF TEMP-TABLE TTlot NO-UNDO
    BEFORE-TABLE biTTlot
    FIELD lotid          AS INT
    FIELD lotName        AS CHAR
    INDEX lotid IS PRIMARY UNIQUE lotid
    INDEX lotname IS UNIQUE lotname.
DEF DATASET dsLot FOR TTlot.
DEF VAR hProc AS HANDLE NO-UNDO.

RUN examples/refill2.p PERSISTENT SET hProc.
MESSAGE 'Before 1st Call' CAN-FIND(FIRST TTlot) dataset
dsLot:handle
     VIEW-AS ALERT-BOX.
RUN getData IN hProc (OUTPUT DATASET dsLot /*BY-REFERENCE*/).
MESSAGE 'After 1st call.  TTlot avail:' CAN-FIND(FIRST TTlot)
dataset dsLot:handle
     VIEW-AS ALERT-BOX.
RUN getData IN hProc (OUTPUT DATASET dsLot /*BY-REFERENCE*/).

/* examples/refill2.p */
DEFINE TEMP-TABLE TTlot NO-UNDO /*REFERENCE-ONLY*/
    BEFORE-TABLE biTTlot
    FIELD lotid          AS INT
    FIELD lotName        AS CHAR
    INDEX lotid IS PRIMARY UNIQUE lotid
    INDEX lotname IS UNIQUE lotname.
DEF DATASET dsLot /*REFERENCE-ONLY*/ FOR TTlot.
DEF VAR hDs   AS HANDLE NO-UNDO.

hDs = DATASET dsLot:HANDLE.
PROCEDURE getData:
    DEF OUTPUT PARAMETER DATASET FOR dsLot.

    MESSAGE 'In refill2' CAN-FIND(FIRST TTlot) dataset
dsLot:handle
         VIEW-AS ALERT-BOX.
    CREATE TTlot.
    ASSIGN TTlot.lotid          = 1
           TTlot.lotName        = 'L123'.
END.
```

9. Fill associated tables

There are a couple of instances where there may not be a fully populated dataset with all associated tables.

The first example is when row changes are sent from client to server. Normal practice is to use GET-CHANGES to send minimal data of only changed rows across the wire. However during validation you may need access to associated tables (assuming that you are separating your business logic from data access). Associated tables may be “children” or “parents”.

In the second example you may have populated a table in one service and now need associated data populate by another service. A dataset fill is triggered from the top down, so if the parent buffer is populated it does not trigger the population of child/associated rows.

In another instance you may wish to refresh associated rows after save-row-changes.

In the first instance you can always ensure that parent rows are sent to the server by using parent mode with GET-CHANGES. However parent mode requires that you use a dataset prefix and I my instance I didn’t want that. To avoid that I used an intermediate dataset but then found that I had to set the ORIGIN-HANDLE. The following is an example that checks a context if parents are populated.

```
    CREATE DATASET hDSChanges.
    hDSChanges:CREATE-LIKE(phDataSet).
    IF CAN-FIND(FIRST ttContext WHERE ttContext.ContextName  = 
"getParents"
                                        AND CAN-
DO('t*,y*',ttContext.ContextValue)) THEN DO:
        CREATE DATASET hDSparent.
        /* need to add a prefix otherwise we get error 12750 on 
get-changes  */
        hDSparent:CREATE-LIKE(phDataSet,'t_').
        hDSparent:GET-CHANGES(phDataSet,YES).
        hDSChanges:COPY-DATASET(hDSparent).
        DO iBuffer = 1 TO hDSparent:NUM-BUFFERS:
            hDSChanges:GET-BUFFER-HANDLE(iBuffer):TABLE-
HANDLE:ORIGIN-HANDLE = hDSparent:GET-BUFFER-HANDLE(iBuffer):TABLE-
HANDLE:ORIGIN-HANDLE.
        END.
        DELETE OBJECT hDSparent NO-ERROR.
    END.
    ELSE
        hDSChanges:GET-CHANGES(phDataSet).
```

One issue with the above method is that you don’t really know how long the client had the parent rows and whether are up-to-date. It is usual to have the before buffer checked for current changed, however, the unchanged rows are not checked. So there are still two other potential issues - ensuring the parent rows are up-to-date and populating child/associated rows.

The solution to this can also be used to populate associated rows in the other examples described above – at least from the top relation down and a similar technique can be employed to populate up or to begin population further down the tree (although relations will have be adjusted and highlighted earlier).

Example code is in the appendix labelled refreshDataset. The basic principle is

Create a temporary dataset copy like the dataset to be refreshed
Find the top level relation
Get the parent buffer of that relation
Establish the parent key
Loop through the parent creating a fill-where-string and filling copy
If refresh, loop through unchanged buffers and update from copy
Copy dataset using append to get missing buffers

10. Use SELF to identify the TT buffer in a fill row callback

To identify the reference TT buffer handle of the callback buffer in a callback procedure use the self handle as show below i.e. SELF in this context is equivalent to phDataSet:GET-BUFFER-HANDLE(<TTname>). This is particularly useful where the callback procedure as in the following example uses standard callback code.

```
PROCEDURE adfAfterRowFill :
    DEF INPUT PARAMETER DATASET-HANDLE phDataSet.

    DEF VAR hTTbuffer         AS HANDLE NO-UNDO.

    hTTbuffer = SELF.
```

In a specific row callback for a TT, the TT buffer will be available and you can just reference it. The database buffer will also be available (unless it was deleted) and may be referred as defined by the data-source. If the default database buffer was not used then you will have to reference the buffer dynamically or re-find it from the TT buffer.

11. Try and keep your dataset code non-dataset specific

Just about all of my dataset use is within a service oriented environment. As such a service should perform all the operations related to the service and at the basic level that includes all fetches and saves. For example all customer fetches and saves should be done by the customer service. This sounds basic and simple but it causes a number of issues. As was demonstrated earlier you can use the 'FOR <dataset> as a parameter to facilitate 4GL coding, but this then restricts the service to that dataset. You could create a customer dataset each time you want to operate with the customer service but this is not as simple as it seems. For example on a save you will have to

(1) Create a customer dataset which assumes a definition available
(2) Copy the customer TT to the customer dataset
(3) Run a save in the customer service
(4) Delete the local customer TT assuming it saved
(5) Copy to customer TT from the customer dataset
(6) Delete the created customer dataset
(7) Handle any context and message issues

Wouldn't it be much easier for the customer service to handle the customer TT within any given dataset? There are still coding issues but at least there is no overhead in copying the TT data as the dataset is passed by reference.

The main issues to be solved in allowing the customer service to handle all customer processing is -

(1) Set context so the customer service only uses customer tables
(2) Handle any context and message conflict issues
(3) Ensure the customer service can process any dataset

The first two issues are framework specific and the third is easily solved by the lessons we have already learnt about datasets.

We could of course just use dynamic programming and if the customer TT is not consistent with the standard customer service TT (or whatever temp-table and service that we are using), then that is what we'd have to do.

Firstly, don't assume the temp-table buffer names are consistent between datasets. They may be named differently as they have a different use or they may have an extension or prefix to allow for multiple instances with the one object or service. In call-backs always use SELF as described earlier to get the handle to the TT the invoked the call-back.

Secondly, using ‘FOR <dataset>’ facilitates 4GL coding but restricts the procedure to the specified dataset format. So how do we receive any dataset and process specific tables using 4GL coding?

Just as a prodataset can be passed by reference, so a temp-table can be passed by reference, and as you can pass a dataset or it’s handle and receive it either way, the same is true of temp-tables.

The principle is this. Receive the unknown dataset as a dataset-handle and dynamically find the temp-tables for the service and pass them to another internal procedure by-reference. In the called procedure use input parameters ‘FOR <TTname>’ and then you can code using standard 4GL coding.

```
PROCEDURE ttCustomerAfterRowFill:
    DEF INPUT PARAMETER DATASET-HANDLE phDataSet.

    DEF VAR hTT AS HANDLE NO-UNDO.

    hTT = SELF:TABLE-HANDLE.
    RUN ttCustomerARF (TABLE-HANDLE hTT BY-REFERENCE).
END PROCEDURE.

PROCEDURE ttCustomerARF:
    DEF INPUT PARAMETER TABLE FOR ttCustomer.

    FIND Customer WHERE Customer.Customer-no =
ttCustomer.Customer-no NO-LOCK.

    ttCustomer.hasActivity  = DYNAMIC-
FUNCTION('hasActivity',Customer.Customer-no)
        ttCustomer.dbRowid      = ROWID(Customer).
END PROCEDURE.
```

Currently there is no call-back on saves and your framework must provide the equivalent of a call-back. A most of the demonstration frameworks pass the dataset handle to the save “call-back” and make the temp-table buffer available. But as I couldn’t use the “for <dataset>” syntax I had the framework assign the buffer handle to the dataset private-data. An example save call-back follows. While this example assumes a handle to the after buffer a future Progress save call-back is likely to assign the before buffer as SELF.

```
PROCEDURE ttCustomerCreateBeginTrans:
    DEF INPUT PARAMETER DATASET-HANDLE phDataSet.

    DEF VAR hTT AS HANDLE NO-UNDO.

    ASSIGN
        hTT = WIDGET-HANDLE(phDataSet:PRIVATE-DATA)
        hTT = hTT:TABLE-HANDLE.

    RUN ttCustomerCBT (TABLE-HANDLE hTT BY-REFERENCE).
END PROCEDURE.
```

A function in the appendix may be used to pass up to 5 temp-tables (which can be increased) to a named procedure and a call example follows. The function allows for table name extensions.

```
DYNAMIC-FUNCTION('runWithTT',
                 'ttCustomer,ttProperty', /* table list */
                 'ttCustomerARF',         /* run proc   */
                 DATASET-HANDLE phDataSet BY-REFERENCE).
```

But what want to pass other parameters in addition to the temp-tables? The function runWithTT calls another function getTTHandles() to get the table handles and that can be used to write your own call.

There are also circumstances where the temp-table names may vary with datasets. In this instance a case statement may examine the received dataset as below.

```
      CASE ENTRY(1,phDataSet:NAME,'_'):
        WHEN 'dsLodgement' THEN
            DYNAMIC-FUNCTION('runWithTT', 'ttLodgement,ttProperty',
'ttLodgementCBT', DATASET-HANDLE phDataSet BY-REFERENCE).
        WHEN 'dsApplication' THEN
            DYNAMIC-FUNCTION('runWithTT', 'ttApplication,ttProperty',
'ttLodgementCBT', DATASET-HANDLE phDataSet BY-REFERENCE).
        OTHERWISE
            DO:
                MESSAGE 'Place error code here'.
                RETURN.
            END.
      END CASE.
```

12. INPUT/OUTPUT has little meaning when passed by reference

In the example in the previous tip, change the parameter to INPUT and pass BY-REFERENCE. When you run you will notice that the dataset is still populated and returned. There is an error on the second create, but that is because I am duplicating the data. The default fill mode of merge raises no error. When passing BY-REFERENCE or BIND you are essentially passing a handle and INPUT/OUTPUT doesn't apply. You may choose to use OUTPUT or INPUT-OUTPUT for documentation. However, once you set the parameter in the procedure or function, then you must use it consistently in the defined way or get a compiler error.

13. Remember to set and unset tracking-changes even on the server

A common problem is forgetting to set tracking-changes when making changes to the after-image TT. Setting tracking-changes seems a natural operation when collecting screen changes, but on the server it can often be forgotten. There may be a valid reason to not capture the change (such as populating TT records that are to be changed as described below), but if you want the change to be saved then set TRACKING-CHANGES to TRUE before the change and TRACKING-CHANGES to FALSE after the change is done.

TRACKING-CHANGES is responsible to create the before buffer and assign the ROW-STATE. In reality once these are set then you can capture a change without setting TRACKING-CHANGES, however, it good practice to ensure changes are captured and provides good documentation.

TRACKING-CHANGES is an attribute of the TT and an earlier example showed how that was obtained e.g. `BUFFER ttCustomer:TABLE-HANDLE.`

Ensure TRACKING-CHANGES is FALSE before GET-CHANGES, FILL, SAVE-ROW-CHANGES, and EMPTY-DATASET.

At times TRACKING-CHANGES should be turned off. For example when wanting to record changes to a record that is not currently populated you must.

(1) set TRACKING-CHANGES to FALSE
(2) CREATE and BUFFER-COPY to the PDS TT
(3) set TRACKING-CHANGES to TRUE
(4) make the desired TT changes
(5) set TRACKING-CHANGES to FALSE
(6) SAVE-ROW-CHANGES

Failing to ensure TRACKING-CHANGES are set to false before the TT create will result in an add or at least an attempted add at SAVE-ROW-CHANGES.

14. Don’t forget to accept or reject changes

If you use a framework when the client receives a saved dataset it will usually accept or reject changes. However, there may be circumstances where you need to ensure changes are accepted at other times.

As an example on a fetch a transaction log was saved on the server using the standard save changes technique. The dataset was set to the client changes were recorded and the log transaction updated. However, when the dataset was set to the server the save changes attempted to re-add the log instead of updating it. The reason is that the original add tracking changes was sent from the server to the client, not accepted as it was a fetch, and then the other changes and the log add was sent to the server. The log was really a committed transaction when first added and needed to be accepted.

15. Create an anyChange function to check if get-changes had changes

When saving changes a change PDS is usually created and populated by GET-CHANGES. However if the user actually made no change the change PDS will be sent to the server with no data. Write a function as below to check if there are any changes to sent to the server and execute with the change PDS as input.

```
FUNCTION anyChange RETURNS LOGICAL
  ( hChangeDS AS HANDLE ) :
    DEF VAR iBuffer AS INT    NO-UNDO.
    DEF VAR hBuffer AS HANDLE NO-UNDO.

    DO iBuffer = 1 TO hChangeDS:NUM-BUFFERS:
        hBuffer = hChangeDS:GET-BUFFER-HANDLE(iBuffer).
        hBuffer = hBuffer:BEFORE-BUFFER.
        IF NOT VALID-HANDLE(hBuffer) THEN
            NEXT.
        hBuffer:FIND-FIRST('',NO-LOCK) NO-ERROR.
        IF hBuffer:AVAIL THEN
            RETURN YES.
    END.
    RETURN NO.
END FUNCTION.
```

16. Use NO-UNDO TT's when saving row changes

When a dataset is saved and an error occurs, transactions are undone including TT buffer errors and error string attributes. The default attribute of a dynamic TT is NO-UNDO but if you use a static TT you need to specify NO-UNDO. In the following example the ERROR-STRING displays ? and ERROR is 'No' despite the lotid 1 duplication on the DB table primary unique key. With NO-UNDO in place the ERROR-STRING and ERROR display correctly. Also notice that the after and before buffer ERROR and ERROR-STRING are consistent.

```
DEF TEMP-TABLE TTlot /*NO-UNDO*/
   BEFORE-TABLE biTTlot
   FIELD lotid        AS INT
   FIELD lotName      AS CHAR
   INDEX lotname IS PRIMARY UNIQUE lotname.
DEF DATASET dsLot FOR TTlot.

DEF TEMP-TABLE lot
   FIELD lotid        AS INT
   FIELD lotName      AS CHAR
   INDEX lotid IS PRIMARY UNIQUE lotid.

TEMP-TABLE TTlot:TRACKING-CHANGES = YES.
CREATE TTlot.
ASSIGN TTlot.lotid       = 1
     TTlot.lotName      = 'L123'.
CREATE TTlot.
ASSIGN TTlot.lotid       = 1
     TTlot.lotName      = 'L998'.
TEMP-TABLE TTlot:TRACKING-CHANGES = NO.

DEFINE DATA-SOURCE srcLot FOR lot.
BUFFER TTlot:ATTACH-DATA-SOURCE(DATA-SOURCE
srcLot:HANDLE).
outer-block: DO TRANSACTION:
   FOR EACH biTTlot:
      BUFFER biTTlot:SAVE-ROW-CHANGES() NO-ERROR.
      IF BUFFER biTTlot:error THEN DO:
         BUFFER biTTlot:ERROR-STRING = ERROR-STATUS:GET-
MESSAGE(1).
         LEAVE.
      END.
   END.
   IF DATASET dsLot:ERROR THEN UNDO outer-block, LEAVE outer-block.
END.
FOR EACH TTlot:
   DISP TTlot.lotid BUFFER TTlot:ERROR BUFFER TTlot:ERROR-STRING
FORMAT 'x(40)' dataset dsLot:ERROR.
END.
FOR EACH biTTlot:
   DISP biTTlot.lotid BUFFER biTTlot:ERROR BUFFER biTTlot:ERROR-
STRING FORMAT 'x(40)' dataset dsLot:ERROR.
END.
FOR EACH lot:
   DISP lot.
END.
```

17. Saving LOBS

As at OE10.1C save-row-changes does not support lobs i.e. clob's and blob's. It is not the actual save that is the issue but the validation of the BI buffer to determine save conflicts. To save a row with lobs set the third parameter of the save-row-changes to true and perform your own save lob save. The following is an example from a framework.

```
      DO iBufferIndex = 1 TO hDataSource:NUM-SOURCE-BUFFERS :
          hBeforeBuff:SAVE-ROW-CHANGES(iBufferIndex,'',YES) NO-ERROR.
/* YES = NO-LOBS */
          /* capture any errors in the for buffer Error-string */
          IF hBeforeBuff:ERROR THEN DO:
            IF ERROR-STATUS:NUM-MESSAGES > 0 THEN
              DO iErrors = 1 TO ERROR-STATUS:NUM-MESSAGES:
                  hBeforeBuff:ERROR-STRING = (IF hBeforeBuff:ERROR-
STRING = ? THEN "" ELSE hBeforeBuff:ERROR-STRING)
                        + STRING(ERROR-STATUS:GET-NUMBER(iErrors)) + ":
":U
                        + ERROR-STATUS:GET-MESSAGE(iErrors) + ": ":U.
/* 6 */
              END.
              ELSE
                hBeforeBuff:ERROR-STRING = (IF hBeforeBuff:ERROR-STRING
= ? THEN "" ELSE hBeforeBuff:ERROR-STRING)
                         + "Your request could not be processed".
              UNDO BLOCK, LEAVE BLOCK. /* abort all the updates and
leave */
          END.
          IF hBeforeBuff:HAS-LOBS AND NOT
saveLobs(hBeforeBuff,phBuffer,hDataSource:GET-SOURCE-
BUFFER(iBufferIndex)) THEN
            UNDO BLOCK, LEAVE BLOCK. /* abort all the updates and leave
*/
      END.
```

The function call saveLobs is in the appendix. This code simply save the lobs and has no code to compare and error where the BI is different than the current database values.

18. After SAVE-ROW-CHANGES ensure usual after-row-fill completed

After SAVE-ROW-CHANGES ensure that the after buffer is refreshed with the latest values for calculated fields. A standard row save will not refresh the after buffer for fields that are not populated by a data-source. You may have calculated fields, and in my case as discussed earlier I capture the DB rowid and the fill time on every row.

19. Capture user changes after a save conflict

When rows changes have been applied the before and after buffers are updated and the buffers are also updated with the latest database values when save-row-changes detects that the before-row buffer values on changed fields is different than the current database values i.e. another user has made changes since the current user retrieved their copy of the record to update.

Now I wanted to advise the user what they originally saw, what they entered, and what the current database value is and allow them to choose between them or cancel out. The dataset returned only included the current values and I didn't want to save values on the off chance there was a conflict. I was intending to pass back these details from the server, but I realised that I could resolve the issue at the client.

When a dataset is sent from the client, the standard practice is to create a dataset like the one changed and then use get-changes to populate the created dataset with only the records that have changed. The dataset that returns is an updated version of the change dataset and therefore the original dataset is available with the before and after images.

The actual coding of this solution is very framework specific and so is not presented here.

20. Consider you save methodology and your transaction scope

As of 10.1C the way in which you do the save is left to you. You may have been given some techniques in training or have downloaded auto-edge. At the time of writing the standard auto-edge only caters for separate transactions and has no model for saving as a single transaction. John Sadd's white paper ***"Implementing the OpenEdge Reference Architecture: 7 Advanced Business Language"*** has a discussion on this and other topics and can be downloaded from Progress's White Papers.

You will also find that you may traverse straight through the buffers or from the top level buffers down.

Before you blindly accept any save methodology you are encouraged to thoroughly test it. In a save-row-changes example error-status:get-message(n)'s where recorded in the appserver log. Adding no-error prevented the log messages but it prevented transaction undo and an undo had to be added at the appropriate place.

You need to consider how you handle error messages. Do you want to traverse buffers and rows to collect all errors or do you collect them when they occur and pass them back as 1 string or in a separate message and/or error TT?

When you get errors back to the client how do you identify errors - by error messages or by dataset:error etc. Test what you get back when you do get an error – how do you refresh you buffers – what has happened to the data the user entered – how is this affected when there is a current-changed conflict i.e. what does the user get now and how do you proceed.

These issues are another large topic. At this juncture you are encouraged to think on your processing, explore and test until you have it working satisfactorily.

21. Develop a method to copy between datasets

If you do need to copy one dataset to another and the datasets do not match you may be forced to use a loose copy. However a loose copy will only copy fields where the names are the same. If this is not the case then you can define the TT in one dataset as the data-source of the other and use the field pairs list to copy fields that are the same but named differently.

The loose copy method in the appendix allows for a number of copy variations.

APPENDIX EXAMPLES

Empty Dataset

```
PROCEDURE emptyDataset :
    DEFINE INPUT PARAMETER DATASET-HANDLE  hDataSet.

    DEF VAR iBuffer AS INT NO-UNDO.

    DO iBuffer = 1 TO hDataSet:NUM-BUFFERS:
        hDataSet:GET-BUFFER-HANDLE(iBuffer):TABLE-HANDLE:TRACKING-
CHANGES  = NO.
    END.
    hDataSet:REJECT-CHANGES().
    hDataSet:EMPTY-DATASET().
END PROCEDURE.
```

RunWithTT

```
FUNCTION runWithTT RETURNS LOGICAL
      ( INPUT cTTs  AS CHAR, INPUT cProc AS CHAR, INPUT DATASET-HANDLE
hDs ):

    DEF VAR hTT AS HANDLE EXTENT NO-UNDO.
    EXTENT(hTT) = NUM-ENTRIES(cTTs).

    RUN getTTHandles IN THIS-PROCEDURE (cTTs, DATASET-HANDLE hDs BY-
REFERENCE, OUTPUT hTT) NO-ERROR.
    IF ERROR-STATUS:ERROR THEN DO:
        MESSAGE '## runWithTT error ' RETURN-VALUE.
        RETURN ERROR-STATUS:ERROR.
    END.

    CASE EXTENT(hTT):
        WHEN 1 THEN RUN VALUE(cProc) IN TARGET-PROCEDURE (TABLE-HANDLE
hTT[1] BY-REFERENCE) NO-ERROR.
        WHEN 2 THEN RUN VALUE(cProc) IN TARGET-PROCEDURE (TABLE-HANDLE
hTT[1] BY-REFERENCE,TABLE-HANDLE hTT[2] BY-REFERENCE) NO-ERROR.
        WHEN 3 THEN RUN VALUE(cProc) IN TARGET-PROCEDURE (TABLE-HANDLE
hTT[1] BY-REFERENCE,TABLE-HANDLE hTT[2] BY-REFERENCE,TABLE-HANDLE
hTT[3] BY-REFERENCE) NO-ERROR.
        WHEN 4 THEN RUN VALUE(cProc) IN TARGET-PROCEDURE (TABLE-HANDLE
hTT[1] BY-REFERENCE,TABLE-HANDLE hTT[2] BY-REFERENCE,TABLE-HANDLE
hTT[3] BY-REFERENCE,TABLE-HANDLE hTT[4] BY-REFERENCE) NO-ERROR.
        WHEN 5 THEN RUN VALUE(cProc) IN TARGET-PROCEDURE (TABLE-HANDLE
hTT[1] BY-REFERENCE,TABLE-HANDLE hTT[2] BY-REFERENCE,TABLE-HANDLE
hTT[3] BY-REFERENCE,TABLE-HANDLE hTT[4] BY-REFERENCE,TABLE-HANDLE
hTT[5] BY-REFERENCE) NO-ERROR.
    END.
    RETURN ERROR-STATUS:ERROR.

END FUNCTION.

FUNCTION getTThandle RETURNS HANDLE
      ( INPUT cTT  AS CHAR, INPUT DATASET-HANDLE hDs ):

    DEF VAR hTT     AS HANDLE NO-UNDO.
    DEF VAR iNo     AS INT    NO-UNDO.

    hTT = hDs:GET-BUFFER-HANDLE(cTT):TABLE-HANDLE NO-ERROR.
    IF VALID-HANDLE(hTT) THEN
        RETURN hTT.
    DO iNo = 1 TO hDs:NUM-BUFFERS:
        hTT = hDs:GET-BUFFER-HANDLE(iNo):TABLE-HANDLE NO-ERROR.
        IF ENTRY(1,hTT:NAME ,'_') = cTT THEN
            RETURN hTT.
    END.
    RETURN ?.
END FUNCTION.
```

Loose Copy Dataset

```
PROCEDURE copyDataset:
    /* Procedure than copies 1 dataset to another using a loose copy
*/
    /* Note that all fields are comma separated and a semi colon
separates tables */
    /*       within each pairs and fields list below
*/
    /* cOptions are -
*/
    /* empty-temp-Table - empty the TT''s listed in cTables output ds
before copy */
    /* empty-dataset    - empty the output ds before copy
*/
    /* append-mode, replace-mode, current-only copy method options
*/

    CREATE WIDGET-POOL.

    DEF INPUT PARAMETER cOptions       AS CHAR   NO-UNDO.
    DEF INPUT PARAMETER cTablePairs    AS CHAR   NO-UNDO.
    DEF INPUT PARAMETER cFieldPairs    AS CHAR   NO-UNDO.
    DEF INPUT PARAMETER cExceptFields  AS CHAR   NO-UNDO.
    DEF INPUT PARAMETER cIncludeFields AS CHAR   NO-UNDO.
    DEF INPUT PARAMETER DATASET-HANDLE hDSin.
    DEF INPUT PARAMETER DATASET-HANDLE hDSout.

    DEF VAR hSource    AS HANDLE NO-UNDO.
    DEF VAR hBufIn     AS HANDLE NO-UNDO.
    DEF VAR hBufOut    AS HANDLE NO-UNDO.
    DEF VAR iNo        AS INT    NO-UNDO.
    DEF VAR iMax       AS INT    NO-UNDO.
    DEF VAR cTablePair AS CHAR   NO-UNDO.
    DEF VAR cFieldPair AS CHAR   NO-UNDO.
    DEF VAR cExcept    AS CHAR   NO-UNDO.
    DEF VAR cInclude   AS CHAR   NO-UNDO.

    ASSIGN
        cFieldPairs    = TRIM(cFieldPairs,'')
        cExceptFields  = TRIM(cExceptFields,'')
        cIncludeFields = TRIM(cIncludeFields,'').
    IF NUM-ENTRIES(cFieldPairs,';') > NUM-ENTRIES(cTablePairs,';') THEN
        RETURN 'The number Field Pair Lists must not be greater than
Table Pair Lists'.
    IF NUM-ENTRIES(cExceptFields,';') > NUM-ENTRIES(cTablePairs,';')
THEN
        RETURN 'The number Except Field Lists must not be greater than
Table Pair Lists'.
    IF NUM-ENTRIES(cIncludeFields,';') > NUM-ENTRIES(cTablePairs,';')
THEN
        RETURN 'The number Include Field Lists must not be greater than
Table Pair Lists'.

    IF CAN-DO(cOptions,'empty-temp-table') THEN
        DO iNo = 1 TO NUM-ENTRIES(cTablePairs,';'):
            cTablePair = ENTRY(iNo,cTablePairs,';').
            hBufOut = hDSout:GET-BUFFER-HANDLE(ENTRY(NUM-
ENTRIES(cTablePair),cTablePair)).
            IF NOT VALID-HANDLE(hBufOut) THEN
                RETURN "Invalid table " + ENTRY(NUM-
ENTRIES(cTablePair),cTablePair) + " for output dataset".
            hBufOut:TABLE-HANDLE:TRACKING-CHANGES  = NO.
            hBufOut:REJECT-CHANGES().
            hBufOut:EMPTY-TEMP-TABLE().
        END.
    ELSE
        IF CAN-DO(cOptions,'empty-dataset') THEN
            RUN emptyDataset (DATASET-HANDLE hDSout).

    iMax = max(NUM-ENTRIES(cFieldPairs,';'),NUM-
ENTRIES(cExceptFields,';'),NUM-ENTRIES(cIncludeFields,';')).
    DO iNo = 1 TO iMax:
        ASSIGN
            cTablePair = ENTRY(iNo,cTablePairs,';')
            cFieldPair = (IF NUM-ENTRIES(cFieldPairs,';') < iNo THEN ''
                          ELSE
                             ENTRY(iNo,cFieldPairs,';'))
            cExcept    = (IF NUM-ENTRIES(cExceptFields,';') < iNo THEN
''
                          ELSE
                             ENTRY(iNo,cExceptFields,';'))
            cInclude   = (IF NUM-ENTRIES(cIncludeFields,';') < iNo THEN
''
                          ELSE
                             ENTRY(iNo,cIncludeFields,';')).
        CREATE DATA-SOURCE hSource.
        hBufIn = hDSin:GET-BUFFER-HANDLE(ENTRY(1,cTablePair)).
        IF NOT VALID-HANDLE(hBufIn) THEN
            RETURN "Invalid table " + ENTRY(1,cTablePair) + " for input
dataset".
        hSource:ADD-SOURCE-BUFFER(hBufIn,?).
        hBufOut = hDSout:GET-BUFFER-HANDLE(ENTRY(2,cTablePair)).
        IF NOT VALID-HANDLE(hBufOut) THEN
            RETURN "Invalid table " + ENTRY(2,cTablePair) + " for
output dataset".
        hBufOut:ATTACH-DATA-
SOURCE(hSource,ENTRY(iNo,cFieldPairs,';'),cExcept,cInclude).
    END.

    hDSout:COPY-DATASET(hDSin,CAN-DO(cOptions,'append-mode'),CAN-
DO(cOptions,'replace-mode'),YES,REPLACE(cTablePairs,';',','),CAN-
DO(cOptions,'current-only'),'').
    IF iMax > 0 THEN
        DO iNo = 1 TO hDSout:NUM-BUFFERS:
            hSource = hDSout:GET-BUFFER-HANDLE(iNo).
            hSource:DETACH-DATA-SOURCE NO-ERROR.
            DELETE OBJECT hSource.
        END.
    RETURN ' '.
END.
```

Save Lobs

```
FUNCTION saveLobs RETURNS LOGICAL
    ( hBeforeBuff  AS HANDLE, phBuffer AS HANDLE, phDbBuffer AS HANDLE
):

    DEF VAR iNo           AS INT    NO-UNDO.
    DEF VAR iErrors       AS INT    NO-UNDO.
    DEF VAR hTTField      AS HANDLE NO-UNDO.

    phBuffer:FIND-BY-ROWID(hBeforeBuff:AFTER-ROWID,NO-LOCK) NO-ERROR.
    IF NOT phBuffer:AVAIL THEN DO:
        ASSIGN
            hBeforeBuff:ERROR        = YES
            hBeforeBuff:ERROR-STRING = "Can't find after buffer for " +
hBeforeBuff:NAME.
        RETURN NO.
    END.
    DO TRANSACTION:
        phDbBuffer:FIND-BY-ROWID(phBuffer:DATA-SOURCE-ROWID,EXCLUSIVE-
LOCK) NO-ERROR.
        IF NOT phDbBuffer:AVAIL THEN DO:
            ASSIGN
                hBeforeBuff:ERROR        = YES
                hBeforeBuff:ERROR-STRING = "Can't find database buffer
for " + hBeforeBuff:NAME.
            RETURN NO.
        END.
        DO iNo = 1 TO phBuffer:NUM-FIELDS:
            hTTField = phBuffer:BUFFER-FIELD(iNo).
            IF NOT CAN-DO('clob,blob',hTTField:data-type) THEN
                NEXT.
            phDbBuffer:BUFFER-FIELD(hTTField:name):BUFFER-VALUE =
hTTField:BUFFER-VALUE NO-ERROR.
            IF ERROR-STATUS:ERROR OR ERROR-STATUS:NUM-MESSAGES > 0 THEN
DO:
                hBeforeBuff:ERROR = YES.
                DO iErrors = 1 TO ERROR-STATUS:NUM-MESSAGES:
                    hBeforeBuff:ERROR-STRING = (IF hBeforeBuff:ERROR-
STRING = ? THEN "" ELSE hBeforeBuff:ERROR-STRING)
                    + STRING(ERROR-STATUS:GET-NUMBER(iErrors)) + ": ":U
                    + ERROR-STATUS:GET-MESSAGE(iErrors) + ": ":U. /* 6
*/
                END.
                RETURN NO.
            END.
        END.
        phDbBuffer:BUFFER-RELEASE().
    END.
      RETURN YES.
END FUNCTION.
```

refreshDataset

```
CREATE WIDGET-POOL.

DEF INPUT PARAMETER DATASET-HANDLE phDataSet.

DEF VAR hRelation      AS HANDLE NO-UNDO.
DEF VAR hAfterBuf      AS HANDLE NO-UNDO.
DEF VAR hBeforeBuf     AS HANDLE NO-UNDO.
DEF VAR hKeyField      AS HANDLE NO-UNDO.
DEF VAR hQuery         AS HANDLE NO-UNDO.
DEF VAR cQuery         AS CHAR   NO-UNDO.
DEF VAR cKeyField      AS CHAR   NO-UNDO.
DEF VAR cContextSaved  AS CHAR   NO-UNDO.
DEF VAR cRefreshParent AS CHAR   NO-UNDO.
DEF VAR hNowDs         AS HANDLE NO-UNDO.
DEF VAR hNowBuf        AS HANDLE NO-UNDO.
DEF VAR iBuffer        AS INT    NO-UNDO.
DEF VAR iKey           AS INT    NO-UNDO.
DEF VAR cKey           AS CHAR   NO-UNDO.

CREATE DATASET hNowDs.
hNowDs:CREATE-LIKE(phDataSet,"z").

ASSIGN
    hRelation     = phDataSet:GET-RELATION(1)
    hAfterBuf       = hRelation:PARENT-BUFFER
    cKeyField     = ENTRY(1,hRelation:RELATION-FIELDS)
    hKeyField     = hAfterBuf:BUFFER-FIELD(cKeyField)
    cContextSaved = DYNAMIC-
FUNCTION('storeContext','EmptyDataset,fillWhereString').

CREATE QUERY hQuery.
cQuery = 'FOR EACH ' + hAfterBuf:NAME + ' no-lock'.
hQuery:SET-BUFFERS(hAfterBuf).
hQuery:QUERY-PREPARE(cQuery).
hQuery:QUERY-OPEN().
REPEAT:
    hQuery:GET-NEXT.
    IF hQuery:QUERY-OFF-END THEN
        LEAVE.
    RUN setContext('fillWhereString','WHERE ' + cKeyField + ' = "' +
hKeyField:BUFFER-VALUE + '"','server').
    RUN fetchWhere IN TARGET-PROCEDURE (OUTPUT DATASET-HANDLE hNowDs
BY-REFERENCE).
    IF hNowDs:ERROR THEN
        LEAVE.
    RUN setContext('EmptyDataset','NO','server').
END.
hQuery:QUERY-CLOSE.
RUN restoreContext (cContextSaved).
IF NOT hNowDs:ERROR THEN DO:
    IF DYNAMIC-FUNCTION('anyChange',phdataset) THEN DO:
        RUN getContext (INPUT "refreshParent":U, OUTPUT
cRefreshParent).
        IF cRefreshParent = ? OR CAN-DO('y*,t*',cRefreshParent) THEN
DO:
```

```
            /* update the unchanged records */
            DO iBuffer = 1 TO phDataSet:NUM-BUFFERS:
                ASSIGN
                    hAfterBuf  = phdataset:GET-BUFFER-HANDLE(iBuffer)
                    hBeforeBuf = hAfterBuf:BEFORE-BUFFER
                    hNowBuf    = hNowDs:GET-BUFFER-HANDLE('z' +
hAfterBuf:NAME).
                hQuery:SET-BUFFERS(hNowBuf).
                hQuery:QUERY-PREPARE("FOR EACH " + hNowBuf:NAME).
                hQuery:QUERY-OPEN().
                hQuery:GET-FIRST().
                DO WHILE NOT hQuery:QUERY-OFF-END:
                    cQuery = ''.
                    DO iKey = 1 TO NUM-ENTRIES(hAfterBuf:KEYS):
                        ASSIGN
                            cKey   = ENTRY(iKey,hAfterBuf:KEYS)
                            cQuery = cQuery
                                   + (IF iKey = 1 THEN 'WHERE ' ELSE '
AND ')
                                   + hAfterBuf:NAME + '.' + cKey + ' =
'
                                   + QUOTER(hNowBuf:BUFFER-
FIELD(cKey):BUFFER-VALUE).
                    END.
                    hAfterBuf:FIND-FIRST(cQuery) NO-ERROR.
                    IF hAfterBuf:AVAIL THEN DO:
                        IF hAfterBuf:ROW-STATE = ROW-UNMODIFIED THEN
                            hAfterBuf:BUFFER-COPY(hNowBuf).
                    END.
                    ELSE DO:
                        cQuery = ''.
                        DO iKey = 1 TO NUM-ENTRIES(hAfterBuf:KEYS):
                            ASSIGN
                                cKey   = ENTRY(iKey,hAfterBuf:KEYS)
                                cQuery = cQuery
                                       + (IF iKey = 1 THEN 'WHERE '
ELSE ' AND ')
                                       + hBeforeBuf:NAME + '.' + cKey +
' = '
                                       + QUOTER(hNowBuf:BUFFER-
FIELD(cKey):BUFFER-VALUE).
                        END.
                        hBeforeBuf:FIND-FIRST(cQuery) NO-ERROR.
              /* if avail then I assume it was deleted so don't copy */
                        IF NOT hBeforeBuf:AVAIL THEN
                            hAfterBuf:BUFFER-COPY(hNowBuf).
                    END.
                    hQuery:GET-NEXT().
                END. /* do while not off end */
            END. /* iBuffer loop */
        END.
    END.
    ELSE  /* or replace all */
        phDataSet:COPY-DATASET(hNowDs,NO,YES).
END.
DELETE OBJECT hQuery.
DELETE OBJECT hNowDs.
```