t Using the Datafile Manager

This section explains the procedures to follow when creating a datafile using the Prograph Datafile Manager. Please note that information about specific Datafile Manager primitives can be found at the end of this chapter. *475*

High Level Overview *476*

The following diagram shows the basic steps involved in creating and maintaining a datafile using the Prograph Datafile Manager. Each step is explained in detail below:

CAUTION:
Table, and Key names are case-sensitive.

Loading Datafile primitives *476*

Datafile primitives are stored in the Datafile Primitives file. This file is located in the Disabled Primitives folder, a subfolder within the Prograph Extensions folder. These primitives are not initially loaded with Prograph Classic, as packaged. In order to use the datafile primitives, shut down Prograph, move the

Datafile Primitives file into the Prograph Extensions folder, and add 200K to Prograph's minimum memory allocation using the Get Info dialog. The datafile primitives will be available the next time you start Prograph.

Creating A New Datafile "477"

Every datafile must contain at least one table. Likewise, tables contain clusters which are indexed by keys. Therefore, creating a new datafile involves the following steps:

1.
Create a datafile.

2.
Create a table (at least one) in the datafile.

3.
Create a key for the table (a table can have several keys).

These three steps are described in detail in the following sections.

How to Create a Datafile "477"

Datafiles are created by calling the db-new primitive. This primitive must be passed the name of the data file you wish to create. db-new creates a new data file on your disk and return an error number (indicating whether or not the primitive succeeded) and an id number which you will need later for creating tables within the datafile. The method below shows how to create a datafile called My Database:

he DB id# parameter returned by db-new is needed later for creating tables and for closing the data file, therefore you should store the DB id# so that you may use it when necessary. The example above stores the DB id# in a persistent.

Note that the filename supplied to the primitive db-new (“My database”) in the previous example could also contain the path of the file (e.g. “HD80:Datafolder:My Database”).

How to Create a Table "478"

Tables are created by calling the table-new primitive. This primitive must be supplied with the following:

r the database file id (DB id# from previous example), and

r a name for the table.

table-new will create a new table within a datafile, and return an error # (indicating whether or not the primitive succeeded) and a table id# which you will need later for creating keys and writing clusters within the table. The following method shows how to create a table called Invoices:

ow to Create a Key for the Table *478*

Keys are created by calling the key-new primitive. This primitive must be supplied with the following:

r the table id (Table id# from the previous example), and

r a name for the key.

key-new will create a new key for a table and return an error #(indicating whether or not the primitive succeeded) and a key id #. The following method shows how to create a key named Customer Number:

_____
IMPORTANT!

Creating a datafile, table or key with the respective primitives (db-new, table-new, or key-new) will also open the item (datafile, table or key) after creating it. However, any newly-created item cannot be used for storing data until it has been closed and then reopened.

For example, while table-new will create a new table within a datafile and then open it, the table id# for this newly created table can only be used for creating new keys – it cannot be used for writing clusters of data to the table. If you want to write data to the table, you must use the table-id# generated by the table-open primitive, as discussed in the next section on Opening Datafiles, Tables, and Keys.

Sample Method for Creating a Datafile *479*

pening Datafiles, Tables & Keys

Opening A Datafile *479*

A datafile may be opened in any of three different modes – Query, Update or Shared. Which mode you choose depends on what you intend to do with the datafile once it is opened.

A datafile opened in Query mode can be read, but new data cannot be written to the datafile. Query mode effectively means read-only mode. Multiple users may simultaneously open and read a datafile in Query mode.

A datafile opened in Update mode may be read and written to by a single user. This user may read data, write new data to the datafile, delete clusters of data or change the structure (tables and keys) of the datafile. Update mode effectively means read-write single-user mode.

A datafile opened in Shared mode can be read or written to by multiple users. A datafile opened in Shared mode can be opened by several users simultaneously and all users may read and write to the datafile. Shared mode effectively means read-write multi-user mode.

Since a datafile contains tables and the tables usually contain keys, the process of opening a datafile involves the following steps:

1.
Open the datafile

2.
Open the table(s) in the datafile

3.
Open the key(s) for the table(s)

How to Open a Datafile *480*

Datafiles are opened by calling the db-open primitive. This primitive must be supplied with the following:

r the mode in which to open the datafile, and

r the name of the datafile that you wish to open.

db-open will open an existing datafile and return an error # (indicating whether or not the primitive succeeded) and an id number which you will need later for opening tables within the datafile. The following method show show to open a datafile named My Database, in Update mode:

he DB id# parameter returned by db-open will be needed later for opening tables and also for closing the datafile, so you should store the DB id# somewhere so that you may use it when necessary. The example above stores the DB id# in a persistent.

How to Open a Table "481"

Tables are opened by calling the table-open primitive. This primitive must be supplied with the following:

r the database id number (DB id# from the previous example), and

r the name of the table.

table-open will open an existing table within a datafile and return an error # (indicating whether or not the primitive succeeded) and a table id# which you will need later for opening keys and reading & writing clusters within the table.

The following method opens a table named Invoices:

he table id# parameter returned by table-open will be needed later for open keys and also for reading and writing clusters in the table, therefore you should store the table id# somewhere so that you may use it when necessary. The example above stores the table id# in a persistent.

How to Open a Key "481"

Keys are opened by calling the key-open primitive. This primitive must be supplied with the following:

r the table id number (Table id# from the previous example), and

r the name of the key.

key-open will open an existing key within a table and return an error # (indicating whether or not the primitive succeeded) and a key id# which you will need later for finding clusters based on key values.The following method opens a key named customer number:

he key id# parameter returned by key-open will be needed later for setting values in the key field and also for finding clusters based on key values, therefore you should store the key id# somewhere so that you may use it when necessary. The example above stores the key id# in a persistent.

The following example opens a datafile in Update mode. The datafile's table and the table's key are also opened. The id numbers associated with each are stored in persistents so that they may be used later. Data can now be read and stored in the datafile's table.

Sample Method for Opening a Datafile "482"

inding/Navigating Clusters "482"

Clusters are the basic data objects stored in a datafile. They are analogous to a single unit of data or a record.

Keys are indexed fields associated with clusters. Key values are the data stored in the key field.

The purpose of a key is to provide a means for indexed, random access to clusters of data within a datafile.

u To locate and access a particular cluster, provide a specific data value to the key-read or key-find primitive in your method, as described in How to Find a Cluster, below.

u To navigate or step through clusters of a datafile, use the key-first, key-last, key- next, and key-previous primitives.

How to Find a Cluster "483"

The Datafile Manager provides several ways to find clusters. Which method is used depends on what you intend to do with the cluster once you've found it.
There are primitives available that allow you to navigate all clusters in a datafile. cluster-first returns the id of the first cluster in the datafile, and cluster-next returns the id of the next cluster.

In many circumstances, you may simply want to read in the cluster's data. In such a case, find the cluster by calling the key-read primitive. This primitive must be supplied with the following:

r the key id number (Key id# from a previous example), and

r the value contained in the desired cluster's key.

The key-read primitive will locate the appropriate cluster (if one exists) and return an error number (indicating whether or not a cluster was found) as well as the cluster's data.

The following method finds a cluster with a key field of 100, and returns the cluster's data:

In some circumstances, you may wish to find the cluster so that you can take some kind of action beyond simply reading the cluster, such as deleting the cluster or overwriting its data. In this case, you must find the cluster and obtain the cluster id number, which is done by using the key-find primitive and the key-value primitive. The key-find primitive must be supplied with the following:

r the key id number (Key id# from a previous example), and

r the value of the key field for the required cluster.

It will locate the appropriate cluster and return an error number (indicating whether or not a cluster was found). Then the key-value primitive is called, to retrieve the cluster id number.

The key-value primitive must be supplied with the key id number (Key id# from a previous example). It reads the key value of the key field and the cluster id number of the cluster.

The following sample method finds a cluster with a key field of 100 (by calling key-find) and then retrieves the cluster id number (by calling key-value):

eading Clusters "484"

How to Read a Cluster

The Datafile Manager provides two ways to read in a cluster's data. The method you use depends on how you intend to select the cluster to be read, as follows:

u To find a cluster based on a key value, use the key-read primitive, which will find and read the cluster. Please refer to the first example in the section, How to Find a Cluster.

u If you are stepping through the clusters sequentially using the key-next, key-previous, key-first or key-last primitives, or if you have already found the desired cluster by some other means, then you can read in the cluster's data by calling the cluster-read primitive.

The cluster-read primitive must be supplied with the following:

r the table id number, and

r the cluster id number

To retrieve a cluster's id number, call the key-value primitive.

cluster-read reads the data stored in a cluster into memory, and returns an error number (indicating whether or not the primitive succeeded) as well as the data of the cluster.

The following method reads in the next cluster:

riting Clusters "485"

Adding a New Cluster to a Table

To add a new cluster to a table within a datafile, call the cluster-write primitive. This primitive must be supplied with the following:

r the table id number (Table id # from the previous example) of the table to which the cluster is to be added,

r the data to be stored within the cluster, and

r a list of key values for the cluster (the key value list should be in alphabetical order by key name);

cluster-write adds a new cluster to a table within a datafile and returns an error number (indicating whether or not the primitive succeeded) and the id number of the new cluster.

The following method adds a new cluster (containing a list of two strings) to a table, and sets the cluster's key field to 100:

verwriting an Existing Cluster "486"

To replace old data stored in a cluster by new data, call the cluster-replace primitive. This primitive must be supplied with the following:

r the table id number,

r the cluster id number, and

r the new data for the cluster.

cluster-replace replaces a cluster within a table and returns an error number (indicating whether or not the primitive succeeded) and the new cluster id number.

The following method replaces an old cluster (that has a key of 100) with a new cluster:

losing Datafiles "486"

Although it is possible to close tables and keys as well as datafiles, it is not usually necessary to close a table or a key, since all tables and keys are closed when a datafile is closed.

How to Close a Datafile

To close a datafile, call the db-close primitive. This primitive must be supplied with the following:

r the id number of the datafile to be closed.

db-close closes a datafile and all of the datafile's tables and keys, and returns an error number to indicate whether or not the primitive succeeded.

The following method closes a datafile: