Writing a Status Routine for High-Level Functions

Both of the two main high-level functions, **DBStartQuery** and **DBGetQueryResults**, call low-level functions repeatedly. After each time they call a low-level function, these high-level functions call a routine that you provide, called a *status routine*. Your status routine can check the result code returned by the low-level function, and can cancel execution of the high-level function before it calls the next low-level function. Your status routine can also update your application's windows after the **DBStartQuery** function has displayed a dialog box.

You provide a pointer to your status routine in the *statusProc* parameter to the **DBStartQuery** and **DBGetQueryResults** functions.

Here is a function declaration for a status routine.

```
pascal Boolean MyStatusFunc (short message, OSErr result,  short dataLen,
    short dataPlaces, short dataFlags, DBType dataType, Ptr dataPtr );
```

Your status routine should return a value of TRUE if you want the **DBStartQuery** or **DBGetQueryResults** function to continue execution, or FALSE to cancel execution of the function. In the latter case, the high-level function returns the userCanceledErr result code.

> **Note:** If you call the **DBStartQuery** or **DBGetQueryResults** function asynchronously, you cannot depend on the A5 register containing a pointer to your application's global variables when the **Data Access Manager** calls your status routine.

The message parameter tells your status routine the current status of the high-level function that called it. The possible values for the message parameter depend on which function called your routine.

The value of the result parameter depends on the value of the message parameter, as summarized in the following list.

Message	Result
kDBUpdateWind	0
kDBAboutToInit	0
kDBInitComplete	Result of **DBInit**
kDBSendComplete	Result of **DBSend**
kDBExecComplete	Result of **DBExec**
kDBStartQueryComplete	Result of **DBStartQuery**
kDBGetItemComplete	Result of **DBGetItem**
kDBGetQueryResultsComplete	Result of **DBGetQueryResults**

The *dataLen, dataPlaces, dataFlags, dataType,* and *dataPtr* parameters are returned only by the **DBGetQueryResults** function, and only when the message parameter equals kDBGetItemComplete. When the **DBGetQueryResults** function calls your status routine with this message, the *dataLen*, *dataPlaces*, and *dataType* parameters contain the length, decimal places, and type of the data item retrieved, and the dataPtr parameter contains a pointer to the data item.

The least significant bit of the *dataFlags* parameter is set to 1 if the data item is in the last column of the row. The third bit of the *dataFlags* parameter is 1 if the data item is NULL. You can use this information, for example, to check the data to see if it meets some criteria of interest to the user, or to display each data item as the **DBGetItem** function receives it. You can use the constants kDBLastColFlag and kDBNullFlag to test for these flag bits.

The **DBGetQueryResults** function returns a results record, which contains a handle to the retrieved data. The address in the *dataPtr* parameter points inside the array specified by this handle. Because the *dataPtr* parameter is not a pointer to a block of memory allocated by the **Memory Manager**, but just a pointer to a location inside such a block, you cannot use this pointer in any **Memory Manager** routines (such as the **GetPtrSize** function). Note also that you cannot rely on this pointer remaining valid after you return control to the **DBGetQueryResults** function.

The following constants can be sent to your status routine in the message parameter by the **DBStartQuery** function:

DBStartQuery message constant	**Meaning**
kDBUpdateWind	The **DBStartQuery** function has just called a query definition function. Your status routine should process any update events that your application has received for its windows.
kDBAboutToInit	The **DBStartQuery** function is about to call the **DBInit** function to initiate a session with a data server. Because initiating the session might involve establishing communication over a network, and because in some circumstances the execution of a query can tie up the user's computer for some length of time, you might want to display a dialog box giving the user the option of canceling execution at this time.
kDBInitComplete	The **DBInit** function has completed execution. When the **DBStartQuery** function calls your status routine with this message, the result parameter contains the result code returned by the **DBInit** function. If the **DBInit** function returns the noErr result code, the **DBStartQuery** function calls the **DBSend** function next. If the **DBInit** function returns any other result code, you can display a dialog box informing the user of the problem before returning control to the **DBStartQuery** function. The **DBStartQuery**

function then returns an error code and stops execution.

kDBSendComplete — The **DBSend** function has completed execution. When the **DBStartQuery** function calls your status routine with this message, the result parameter contains the result code returned by the **DBSend** function. If the **DBSend** function returns the noErr result code, the **DBStartQuery** function calls the **DBExec** function next. If the **DBSend** function returns any other result code, you can display a dialog box informing the user of the problem before returning control to the **DBStartQuery** function. The **DBStartQuery** function then returns an error code and stops execution.

kDBExecComplete — The **DBExec** function has completed execution. When the **DBStartQuery** function calls your status routine with this message, the result parameter contains the result code returned by the **DBExec** function. If the **DBExec** function returns the noErr result code, the **DBStartQuery** function returns control to your application next. If the **DBExec** function returns any other result code, you can display a dialog box informing the user of the problem before returning control to the **DBStartQuery** function. The **DBStartQuery** function then returns an error code and stops execution.

kDBStartQueryComplete — The **DBStartQuery** function has completed execution and is about to return control to your application. The function result is in the result parameter passed to your status routine. Your status routine can use this opportunity to peform any final tasks, such as disposing of memory that it allocated or removing from the screen any dialog box that it displayed.

The following constants can be sent to your status routine in the message parameter by the **DBGetQueryResults** function:

DBGetQueryResults

message constant	Meaning
kDBGetItemComplete	The **DBGetItem** function has completed execution. When the **DBGetQueryResults** function calls your status routine with this message, the result parameter contains the result code returned by the **DBGetItem** function. The **DBGetQueryResults** function also returns values for the *dataLen*, *dataPlaces*, *dataType*, *dataFlags*, and *dataPtr* parameters. For each data item that it retrieves, the **DBGetQueryResults** function calls the **DBGetItem** function twice: once to obtain information about the next data item and once to retrieve the data item. The

DBGetQueryResults function calls your status routine only after calling the **DBGetItem** function to retrieve a data item.

If your status routine returns a function result of FALSE in response to the kDBGetItemComplete message, the results record returned by the **DBGetQueryResults** function to your application contains data through the last full row retrieved.

Data types and results records are described under **Getting Query Results** in the section entitled **Processing Query Results** .

kDBGetQueryResultsComplete — The **DBGetQueryResults** function has completed execution and is about to return control to your application. The function result is in the result parameter passed to your status routine. Your status routine can use this opportunity to peform any final tasks, such as disposing of memory that it allocated or removing from the screen any dialog box that it displayed.

The Listing below shows a status routine for the **DBStartQuery** function. This routine updates the application's windows in response to the kDBUpdateWind message, displays a dialog box giving the user the option of canceling before the data access is initiated, and checks the results of calls to the **DBInit**, **DBSend**, and **DBExec** functions. If one of these functions returns an error, the status routine displays a dialog box describing the error.

```
// A sample status routine
// Assuming inclusion of <MacHeaders>

#include <DatabaseAccess.h>
#include <string.h>

pascal Boolean MyStartStatus (short message, OSErr result, short dataLen,
            short dataPlaces, short dataFlags, DBType dataType,  Ptr dataPtr);
void MyDoActivate (void);
void MyDoUpdate(void);
void MyDisplayDialog(Str255 myString, Boolean *continueQuery);
void DisplayString(Str255 myString);
void DisplayError(Str255 myString, OSErr result);
void MyCleanUpWindows (void);

pascal Boolean MyStartStatus (short message, OSErr result, short dataLen,
                short dataPlaces, short dataFlags, DBType dataType,
                Ptr dataPtr)
{
    Str255     myString;
    Boolean continueQuery;

    continueQuery = TRUE;// Assume user wants to continue with the query
    switch (message) {
    case kDBUpdateWind:
```

```
        // A qdef function has just been called. Find your
        // activate and update events and handle accordingly.

        MyDoActivate();   // Find and handle activate events
        MyDoUpdate();     // Find and handle update events
        break;

    case kDBAboutToInit:

        // About to initiate a session

        // MyDisplayDialog is a routine that displays a dialog box.
        // It takes as input a string you want to display
        // and returns a Boolean telling DBStartQuery
        // whether to continue.
            strcpy((char *) myString, (char *)
                        "\pThe Data Access Manager"
                        "is about to open a session.  This"
                        "could take a while.  Do you want"
                        "to continue?");

        MyDisplayDialog(myString, &continueQuery);
        break;

    case kDBInitComplete:

        // The DBInit function has completed execution.  If there's an
        // error, let the user know what it is.

        if (result)
            switch (result) {
            case rcDBError:
                strcpy((char *) myString, (char *)
                            "\pThe Data Access Manager was "
                            "unable to open the session."
                            "Please check your connections"
                            "and try again later.");

                // DisplayString displays a dialog box containing
                // the string you want to display.
                DisplayString(myString);
                break;

            case rcDBBadDDEV:
                strcpy((char *) myString, (char *)
                    "\pThe Data Access Manager cannot "
                    "find the database extension file it needs in order "
                    "to open a session.  Please check with your system "
                    "administrator to obtain a copy of this file.");
                DisplayString(myString);
                break;

            default:
                strcpy((char *) myString, (char *)
                    "\pThe Data Access Manager was "
```

```
                "unable to open the session.  The error code returned "
                "was");

            // DisplayError displays a dialog box containing
            // the string you want to display plus an error code.
            DisplayError(myString, result);
            break;
        }
    break;

case kDBSendComplete:

    // The DBSend function has completed execution.  If there's an
    // error, let the user know what it is.

    if (result)
        if (result == rcDBError) {
            strcpy((char *) myString, (char *)
                "\pAn error occurred while the Data "
                "Access Manager was trying to send the query.  Please "
                "try again later.");
            DisplayString(myString);
        }
        else {
            strcpy((char *) myString, (char *)
            "\pAn error occurred while the Data "
            "Access Manager was trying to send the query.  The error "
            "code returned was");
            DisplayError(myString, result);
        }
    break;

case kDBExecComplete:

    // The DBExec function has completed execution.  If there's an
    // error, let the user know what it is.

    if (result)
        if (result == rcDBError) {
            strcpy((char *) myString, (char *)
            "\pThe Data Access Manager was "
            "unable to execute the query.  There may be a problem "
            "with the query document or with the database.  Check with "
            "your system administrator.");
            DisplayString(myString);
        }
        else {
            strcpy((char *) myString, (char *)
            "\pAn error occurred while the Data "
            "Access Manager was trying to execute the query.  The error "
            "code returned was");
            DisplayError(myString, result);
        }
    break;
```

```
        case kDBStartQueryComplete:

            // The DBStartQuery function is about to return control
            // to your application.  You can clean up memory
            // and any dialog boxes you left on the screen.
            // MyCleanUpWindows is a routine that does that.

            MyCleanUpWindows();
            break;
        }
        return continueQuery;
}
```