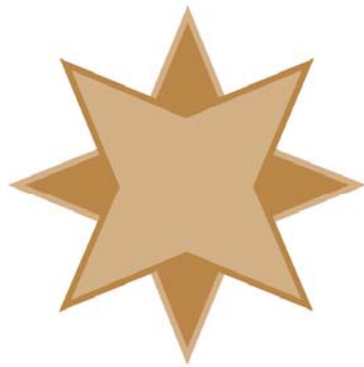# Accessing Pervasive Status Codes From Applications

A White Paper From

GOLDSTAR
SOFTWARE

www.GoldstarSoftware.com

For more information, see our web site at
http://www.goldstarsoftware.com

Accessing Pervasive Status Codes From Applications

Last Updated: 05/26/2009

One of the most frequent types of questions that we get from application users and developers alike is about the various Pervasive Status Codes. Currently, Pervasive has hundreds of distinct status codes, and there is no way for a user (or even a developer) to remember them all. Further, the responsible developer will want to provide lots of good information to the user about a status code so that when an error occurs, the user will be able to use that information to find a solution. The net result is a shorter time to resolution, which should result in a happier user!

Simplest Solution: Nothing

The simplest solution is to do nothing -- just report "Btrieve Error" and blue-screen the machine. If this is your idea of a top-notch "user abusive interface", then I have a drink for you. (Ignore the faint smell of cyanide, it's really very tasty!)

Simple Solution: Use the Documentation

The most obvious solution is to direct the user to the Pervasive documentation. This documentation is already being updated by Pervasive with each new version, and it should always represent the most current information about the database product that the user is running. Further, this documentation has already been compiled and formatted for easy access from a typical end user, and no special processing or coding is required. This combination makes this solution VERY attractive for most developers.

There are, however, several downsides to this solution:

- Users may not know how to get to the Pervasive documentation. This is complicated by the fact that the Documentation used to be a separate item in the Program List, but it is available within PSQLv10 as part of the PCC only. As such, you must provide customized instructions (by PSQL version) telling the user how to find the documentation.

- Users may not have the needed tools or documentation installed. For an ISV creating a compact installation image, these tools and documentation are commonly omitted from the installation, making it impossible for some users to even find the information they need. For a PSQLv10 user without the PCC, the docs are simply unavailable.

- If you install and refer to the PDF files, then you must have a PDF reader available. For most people, this is not an issue, but for servers or some basic applications, PDF readers may not be installed on the machine.

- The interface is far from seamless, and it appears very "kludgey" to the user.

Moderate Solution: Use the Web

Another viable solution that some have experimented with is to provide direct links right to a web site, whether it be a Google Search Window, the Pervasive documentation library site itself, the Pervasive knowledgebase, or even a developer-controlled site. Again, with a minimal amount of coding, you can now provide detailed and up-to-date information about ANY status code directly within a web browser window.

There are still some pitfalls to watch out for:

- Web sites and internet connections may be temporarily unavailable, making the query fail and leaving the user in even more of a quandary then when he started.

- Browser limitations, security configurations, antivirus applications, firewalls, proxy passwords, and other mechanisms commonly used to secure our environments today may block web requests generated from inside an application.

- Web sites can (and do) frequently change structure, making URLs that may work just fine today be completely invalid tomorrow. This can completely date an application and make for a miserable user experience with the dreaded "404 - Page Not Found" error.

- Linking to a search service (like the Pervasive Knowledgebase or a Google window) may provide far too many possible options and unrelated solutions, making it somewhat less than useful to a novice user.

Complicated Solution: Application-Provided Messages

The most complicated solution for developers is to create a customized message system that displays the related text for each specific error code. This is the most complicated solution because it relies on the developer either coding up a lot of static messages or creating some huge look-up table. It is by far the best option for the user since the error message and possible resolutions can be directly integrated into the application, presenting the user with both the error and the possible solutions to that error at the same time.

Like the other solutions, this one also has some major problems:

- There are a LOT of status codes in the Pervasive environment, and it would take a developer a substantial amount of time to hand-code each of these. If the developer skips the deprecated or less common status codes, though, then the random user who DOES get such an odd error is left in the cold.

- Maintaining this information separately from Pervasive seems like a waste of time, and the result is a lot of extra work that's already been done by one developer after another with little expected gain.

- Internationalization of the messages (i.e. translating to different languages) may be difficult for all developers, and will again lead to an extensive amount of work for potentially little value or payback.

Proposed Idea: Application-Provided Messages via a System Table

Goldstar Software would like to propose a solution that could become a standard among Pervasive developers to address the issues presented above and improve both the usability of Pervasive applications as well as the productivity of Pervasive developers. This solution is comprised of a series of system-defined tables that contain the core status codes documentation, along with some basic access methods that expose these tables to the developers who need them. The net result will be a consistent way for developers to access this information, a reduction of the development workload, and almost immediate internationalization of the status codes data for all applications.

The first step to making this happen is creating the system table. The system table can be created very easily from a simple CREATE statement that can be implemented during the same Pervasive System Stored Procedures pre-load process that takes place at installation time. The following SQL statements will suffice:

```
CREATE TABLE PervasiveSysDB.StatusCodesEN (
    StatusCode    INTEGER NOT NULL,
    ShortDescription VARCHAR(250),
    LongDescription LONGVARCHAR
);
CREATE UNIQUE INDEX CodesIndex ON
    PervasiveSysDB.StatusCodesEN(StatusCode);
```

Note that the last two characters on the table name include a language code, EN for English, DE for German, and so on. Most languages already have a two-character abbreviation, so this is a simple extension. By segregating each language into its own file, it will be possible for Pervasive

(or developers) to still keep their installation image to a minimum and yet be completely expandable with new languages as the translations are completed. It is also possible, therefore, to extend this to include a table (say "StatusCodes_AppName") which contains user-defined extensions in which the long description is blank, allowing the developer to provide messages specific to their specific application.

The second step is to populate the table. This can be done with a large set of INSERT statements, or from an import script (just like DEMODATA is populated). The ideal source for this data would be Pervasive Software, as they are already maintaining the primary documentation, and perhaps they can create a simple text extraction from the existing documentation to make this a whole lot easier. Goldstar Software has an initial set of comma-delimited data, provided as a simple extract from the Status Codes Quick Reference Sheet for PSQLv10. You MUST have a license of PSQL to download this document:

   ftp://ftp2.goldstarsoftware.com/statuscodes.csv

This text can then be loaded with the following command:

```
   SQLEXEC DEMODATA "INSERT INTO PervasiveSysDb.StatusCodesEN (StatusCode,
ShortDescription)" /istatuscodes.csv
```

If you do not have SQLExec, get a 30-day trial copy from www.goldstarsoftware.com/tools.asp. Be sure the entire command above is provided on one command line. Note that this import file includes the short description only -- the long text is not yet part of the import. If you do create a larger file (with the long description included), be sure to update the INSERT statement accordingly.

Once you've created and populated the table, it can easily be accessed from any SQL-based or ODBC-based application. To perform a lookup on any given Status Code, use a SQL statement similar to that shown here:

```
   SELECT * FROM PervasiveSysDB.StatusCodesEN WHERE StatusCode = x;
```

Be sure to provide your own status code for the "x" at the end of the statement.

An obvious extension would be to create a System Stored procedure, which accepts as input a status code and a language code (defaulting to "EN") and builds the necessary SQL query for you. Such a stored procedure could look like this:

```
CREATE PROCEDURE psp_status_code (
    in :stat_code   INTEGER,        -- Status Code
    in :lang_code   CHAR(2) = 'EN'  -- Language
```

```
)
RETURNS (
    Status_Code  INTEGER,            -- Status Code
    Short_Description  VARCHAR(250),  -- Short Description
    Long_Description   LONGVARCHAR    -- Long Description
);
BEGIN
    DECLARE :query      LONGVARCHAR;
    -- If the language is null or blank, then use English
    if (:lang_code is null) then
        set : lang_code = 'EN';
    end if;
    if (rtrim(:lang_code) = '') then
        set : lang_code = 'EN';
    end if;
    set :query = 'SELECT * FROM PervasiveSysDB.StatusCodes' + :lang_code +
        ' WHERE StatusCode = ' + :stat_code + ';';
    exec (:query);
END;
```

Of course, SQL access isn't the only access method, and such a table could be accessed directly as a Btrieve file, too. Doing this would be very easy through the use of URI's to open the files. A simple URI could be done like this:

```
btrv://servername/PervasiveSysDb?table=StatusCodesEN
```

Once the Btrieve file is opened, a simple GetEqual on Key 0 with the status code in the key buffer and a 264-byte data buffer will return a Status 0 or 22 (which is ignored) along with the ShortDescription field and the static values from the LongDescription field, namely the Null Indicator Byte (NIB), field offset, and field length. If the LongDescription field is also required, and if the NIB has a value of 0 and if the field length is greater than 0, then the developer can issue a GetPosition call followed by a GetDirect/Chunk to read the LongDescription field directly into his own memory buffer. (Optionally, the code can handle even longer LongDescription fields through the use of GetDirect/Chunk and the Next-In-Record bias.) It would be considered extremely rare to have a long description exceed 60K in size, so it may also be possible to exclude the GetDirect/Chunk logic and simply do a 60K record read, ignoring anything that doesn't fit in that size buffer.

Summary: Does This Solution Really Work?

This type of solution seems to solve most of the major problems on status code reporting, but it is not without costs. Someone, most likely Pervasive Software, would need to provide the initial data build to create the import data. Additionally, if translations were completed, these would again be ideally handled directly at the source, so that when changes are made in the

documentation, they get reflected in the StatusCodes tables as well. It is also possible that two different applications with similar names might conflict, causing naming problems.

Once these costs are covered, though, the benefits for developers will be far reaching. First, it is accessible from all of the standard interface API's, including ODBC (and derivatives), SQL (and derivatives) Btrieve. Second, it could also be accessed via a simple extension to the DTI and DTO interfaces with minimal additional code, and it could even be implemented as a separate Btrieve API call, if appropriate, but it will work within the current interface with no extensions needed, as well. Third, the data is table-driven, meaning that the data can be easily updated, translated, and extended, even on an application-by-application basis if needed. Fourth, as a common (and supported) standard built into a future version of Pervasive PSQL (and maintained by Pervasive), it would offer increased functionality for developers for a limited cost -- and it might help drive engine upgrades if developers knew that this feature was available to use with their applications.

It should also be clear that a solution like this can certainly be implemented by each developer independently, at NO cost to Pervasive. However, if this is done, each developer will be bearing the costs of reinventing this wheel, and the problem will never resolve itself -- it will simply get bigger and bigger as every developer starts to create his own solution and make getting to a common standard in the future a lot more difficult.