Using PERL

DataDirect Connect® for ODBC

Introduction

PERL (Practical Extraction and Report Language) has had strong acceptance in the IT community since the introduction of version 1.0 in 1987. PERL is used widely on UNIX systems, especially Linux, for text manipulation and parsing scenarios that are too intensive for shell commands, but not complex enough for a language like C. Not being able to use shell commands for these actions makes them less portable across different implementations of UNIX. Many organizations today use PERL to handle common tasks associated with web servers and web interfaces to corporate databases. PERL is extremely flexible and has a large support base in the USENET community, which frequently posts “modules” that can be modified for any number of wide ranging tasks. These modules greatly reduce the amount of code a developer needs to implement because they are widely available for anyone's use. This community also offers many resources available to those just beginning to learn PERL.

Because PERL is easily portable across operating systems, several initiatives have been undertaken to develop modules that enable developers to use the ODBC interface, thus enhancing PERL’s portability across corporate databases. DataDirect Connect® *for* ODBC allows PERL applications not only to be completely portable, but also highly scalable. DataDirect Connect *for* ODBC enables applications to achieve better performance through its unique wire protocol technology and network optimizations.

This paper provides information necessary for using DataDirect Connect *for* ODBC with PERL, including configuration instructions and a sample PERL application.

Using DBD::ODBC

DBI is the Database Interface module for the PERL programming language. The DBD::ODBC PERL module is available at http://dbi.perl.org/ and provides a transparent interface to any ODBC-compliant database rather than a particular database. In addition to increasing portability, this module offers PERL developers a wide degree of flexibility when developing PERL applications and insulates the application from any subsequent changes to databases.

After downloading and installing the DBD::ODBC module, you will need to install the DataDirect Connect *for* ODBC drivers and edit the odbc.ini file.

DataDirect TECHNOLOGIES

Installing and Configuring the DataDirect Connect® *for* ODBC Drivers

Download DataDirect Connect *for* ODBC from http://www.datadirect.com. Select DataDirect Connect *for* ODBC from Products on the navigation bar at the top of the main page. Then, click Download on the left side of the product page. You can also view the DataDirect Connect *for* ODBC documentation from the product page, including the installation guide. After downloading the ODBC drivers and documentation, refer to the installation guide to install the drivers.

Creating a Data Source

You will need to create a data source that the PERL application can call to establish a connection to the database. On UNIX, this is done through the system information file, usually named odbc.ini. A sample odbc.ini is part of the DataDirect Connect *for* ODBC installation. Complete instructions on setting environment variables and creating a data source are provided in the *DataDirect Connect for ODBC Reference* that comes with the product.

After completing the procedures for setting up odbc.ini, edit the SQL Server section of the file so that, at a minimum, it contains the following entries:

```
[SQL Server Wire Protocol]
Driver=<odbchome>/lib/ivmsssxx.so
Description=SQL Server
Database= <database_name>
Address= <SQL Server_host, SQL Server_server_port>
QuoteID=No
AnsiNPW=No
```

Where `<odbchome>` is the DataDirect Connect *for* ODBC installation directory; *xx* represents the driver release number; `<database_name>` is the name of the target database; and `<SQL Server_host, SQL Server_server_port>` are the IP address and port of the target database, for example, 120.2.200.176,1433.

Configuration for PERL

Use the following procedure to enable PERL and the DBD::ODBC module to communicate correctly with ODBC drivers on all UNIX platforms. Modification requests have been submitted to the author of the module so that the following procedure should be unnecessary in the future.

1. Set up the environment for ODBC:
    a. Source odbc.sh or odbc.csh in the DataDirect Connect *for* ODBC installation directory to add the ODBC libraries to the shared lib path.
    b. Set the ODBCINI variable to the odbc.ini file.

c. Set the ODBCHOME variable to the DataDirect Connect *for* ODBC installation directory (not the lib directory as specified in the DBD::ODBC documentation).

d. Set the DBI_DSN variable to dbi:ODBC:*data_source_name*, where *data_source_name* is the name of the odbc.ini data source that you plan to use (SQL Server Wire Protocol).

e. Set the DBI_USER variable to the database user.

f. Set the DBI_PASS variable to the database password.

2. Modify Makefile.PL:

   This file contains the following two lines by default:

```
$myodbc = 'intersolve'
  if -f '$odbchome/include/qeodbc.h';
```

   a. You need to relocate these lines so that they precede the line:

```
$myodbc = 'unixodbc'
```

   b. Modify them to read:

```
$myodbc = 'intersolve'
  if -f '$odbchome/include/sqlunx.h';
```

   If you do not move them to precede the `'unixodbc'` entry in the Makefile.PL script, then the script assumes that you are using a generic UNIX ODBC Driver Manager and the Makefile will not generate correctly.

   The file also contains:

```
elsif ($myodbc eq 'intersolve') {
  $opts{DEFINE}  = ""
  print SQLH qq{#include <qeodbc.h>\n};
  if (-f "$odbchome/include/sql.h") {
    print "You seem to have the official header files.\n"
    $opts{INC} .= " -I$odbchome/include"
    print SQLH qq{#include <sql.h>\n#include
      <sqltypes.h>\n#include <sqlext.h>\n};
```

   c. Change this to:

```
elsif ($myodbc eq 'intersolve') {
  $opts{DEFINE}  = ""
  if (-f "$odbchome/include/sql.h") {
    print "You seem to have the official header files.\n"
    $opts{INC} .= " -I$odbchome/include"
    print SQLH qq{#include <sql.h>\n#include
      <sqltypes.h>\n#include <sqlext.h>\n#include
      <sqlucode.h>\n};
```

3. Run the command:

```
perl Makefile.PL
```

   If you receive an error similar to

```
Makefile:90: *** missing separator. Stop.
```

try setting the LANG environment variable to en_US (LANG=en_US).

4. Run the `make` command to compile the DBD::ODBC module source code.
5. Log in as the owner of the PERL installation.
6. Issue `make install` to write the module to the PERL directories.

Sample PERL Application

This simple application (sample_app.pl) enables you to query and enter employee information. The application establishes a connection to a SQL Server database and, based on information entered at the prompts, either queries a table to retrieve employee information or enters employee information into the database. This application is meant to demonstrate in a simplified manner how to connect to a database and enter or retrieve information using ODBC with PERL.

Table Creates and Population

For this application to work, you will need to create a table. In this example, the table name is employee. The Create statement for this table is straightforward:

```
CREATE TABLE employee (LastName char(20),FirstName char(15),EmpID varchar(5),
Office varchar(20), HireDate datetime)
```

The Sample Application

The sample application is named sample_app.pl and can be created in a text editor. This command-prompt application allows you either to query employee information or to enter employee information.

NOTE: You will need to modify the connect string to include a data source that is on your machine (such as SQL Server Wire Protocol) as well as an appropriate user ID and password for your database.

```
use DBI;

  my $dbh = DBI->connect('DBI:ODBC:datasource','userid','pwd')
    or die "Couldn't connect to database:" . DBI->errstr;
  my $sth1 = $dbh->prepare('Select * from employee where LastName = ?')
    or die "Couldn't prepare statement:" . $dbh->errstr;
  my $sth2 = $dbh->prepare('insert into employee
    (LastName,FirstName,EmpID,Office,HireDate) values (?,?,?,?,?)')
    or die "Couldn't prepare statement:" . $dbh->errstr;
  sub trimwhitespace($);
  sub trimwhitespace($)
  {
    my $string = shift;
    string =~ s/^\s+//;
```

```
  $string =~ s/\s+$//;
  return $string;
}

print "Employee Information: \n";
print "Enter 1 to Search Employee Information\n";
print "Enter 2 to Enter Employee Information: >" ;
$AppFn = <STDIN>;
chomp $AppFn;

  if ($AppFn == 1) {
  print "\nEnter Employee last name: >";
  $Last = <STDIN>;
  $Last = trimwhitespace($Last);
  my @data;
  $sth1->execute($Last)
    or die "Couldn't execute the statement:" .$sth1->errstr;
  #READ ONLY THE MATCHING RECORDS AND THEN PRINT THEM OUT
while (@data = $sth1->fetchrow_array()) {
  my $Last = trimwhitespace($data[0]);
  my $First = trimwhitespace($data[1]);
  my $EmpID = trimwhitespace($data[2]);
  my $Office = trimwhitespace($data[3]);
  my $Hire = trimwhitespace($data[4]);
  print "\nLast Name: $Last\nFirstName: $First\nEmployee ID: $EmpID\nOffice:
    $Office\nHire Date: $Hire\n";
}
  if ($sth1->rows == 0) {
    print "No names matched `$Last'.\n\n";
  }

  $sth1->finish;

$dbh->disconnect;
  }

  if ($AppFn == 2){
  print "\nEnter Employee Last Name: >";
  $Last = <STDIN>;
  $Last = trimwhitespace($Last);
  print "\nEnter Employee First Name: >";
  $First = <STDIN>;
  $First = trimwhitespace($First);
  print "\nEnter Employee ID: >";
  $EmpID = <STDIN>;
  $EmpID = trimwhitespace($EmpID);
  print "\nEnter Employee office: >";
  $Office = <STDIN>;
  $Office = trimwhitespace($Office);
  print "\nEnter Employee hire date (yyyy-mm-dd): >";
  $tmpHire = <STDIN>;
  $tmpHire = trimwhitespace($tmpHire);
  $Hire = "{d '$tmpHire'}";
  print $Hire;
  $sth2->execute($Last,$First,$EmpID,$Office,$Hire)
    or die "Couldn't execute the statement:" .$sth2->errstr;
      $sth2->finish;

$dbh->disconnect;
  }
```

Annotation

In this annotation section, the code sample is followed by an explanatory paragraph.

```
use DBI;
```

This code states which modules are used in the sample application. For sample_app.pl, the DBI module enables the application to get information from the SQL Server database through ODBC.

```
my $dbh = DBI->connect('DBI:ODBC:Datasource','userid','pwd')
  or die "Couldn't connect to database:" . DBI->errstr;
```

This is the database connection. The parenthetical statement indicates that DBI will be used for ODBC, and which data source, user ID, and password will be used to connect. You will need to modify the code to use an existing data source that you have created, as well as the user ID and password that you use to connect to the database.

```
my $sth1 = $dbh->prepare('Select * from employee where LastName = ?')
  or die "Couldn't prepare statement:" . $dbh->errstr;
my $sth2 = $dbh->prepare('insert into employee
  (LastName,FirstName,EmpID,Office,HireDate) values (?,?,?,?,?)')
  or die "Couldn't prepare statement:" . $dbh->errstr;
```

These two statements are prepares for the SQL statements that the application uses. The first (`$sth1`) is to retrieve any employee information that matches the input from the user. The second statement (`$sth2`) is to insert the employee data that the user enters from a series of prompts.

```
sub trimwhitespace($);
  sub trimwhitespace($)
  {
    my $string = shift;
    string =~ s/^\s+//;
    $string =~ s/\s+$//;
    return $string;
  }
```

This code declares and defines a subroutine that strips any leading and trailing white space from the variables in the application.

```
print "Employee Information: \n";
print "Enter 1 to Search Employee Information\n";
print "Enter 2 to Enter Employee Information: >" ;
$AppFn = <STDIN>;
trimwhitespace($AppFn);
```

This code prompts the user to enter the function that should be performed, assigns the input to a variable, and strips the variable of any trailing white space.

```
if ($AppFn == 1) {
    print "\nEnter Employee last name: >";
    $Last = <STDIN>;
    $Last = trimwhitespace($Last);
    my @data;
    $sth1->execute($Last)
      or die "Couldn't execute the statement:" .$sth1->errstr;
```

If the user enters `1` to search employee information, this code takes the user input and assigns it to the variable `$Last`. The first prepared statement for selecting employee information (`$sth1`) is then executed with this parameter (`$Last`).

```
  #READ ONLY THE MATCHING RECORDS AND THEN PRINT THEM OUT
while (@data = $sth1->fetchrow_array()) {
  my $Last = trimwhitespace($data[0]);
  my $First = trimwhitespace($data[1]);
  my $EmpID = trimwhitespace($data[2]);
  my $Office = trimwhitespace($data[3]);
  my $Hire = trimwhitespace($data[4]);
  print "\nLast Name: $Last\nFirstName: $First\nEmployee ID: $EmpID\nOffice:
    $Office\nHire Date: $Hire\n";
}
```

This code reads the result set from the array, strips the results of any leading or trailing white space, and prints the results on the screen.

```
  if ($sth1->rows == 0) {
    print "No names matched `$Last'.\n\n";
  }

  $sth1->finish;

$dbh->disconnect;
```

If there are no matching results from the query, this code prints out a message stating that there were no matches. It then drops the connection.

```
  if ($AppFn == 2){
  print "\nEnter Employee Last Name: >";
  $Last = <STDIN>;
  $Last = trimwhitespace($Last);
  print "\nEnter Employee First Name: >";
  $First = <STDIN>;
  $First = trimwhitespace($First);
  print "\nEnter Employee ID: >";
  $EmpID = <STDIN>;
  $EmpID = trimwhitespace($EmpID);
  print "\nEnter Employee office: >";
  $Office = <STDIN>;
  $Office = trimwhitespace($Office);
  print "\nEnter Employee hire date (yyyy-mm-dd): >";
  $tmpHire = <STDIN>;
  $tmpHire = trimwhitespace($tmpHire);
  $Hire = "{d '$tmpHire'}";
  print $Hire;
  $sth2->execute($Last,$First,$EmpID,$Office,$Hire)
    or die "Couldn't execute the statement:" .$sth2->errstr;
      $sth2->finish;

$dbh->disconnect;
  }
```

If the user enters `2` to insert employee information, then this section of code prompts the user to enter the associated employee information, assigns each input to a variable, and strips it of any leading or trailing white spaces. The prepared statement (`$sth2`) to insert the data is then executed with the associated parameters. The statement finishes and the connection is dropped.

Running the Application

Refer to the instructions for executing the application in your particular PERL environment. If you are running WinActive's PERL implementation on Windows, change to the directory where the application resides and enter the following line at the command prompt:

```
perl sample_app.pl
```

The following prompt is displayed, at which point the user types `2` and presses ENTER to insert employee information.

```
Employee Information:
Enter 1 to Search Employee Information
Enter 2 to Enter Employee Information: >2
```

The following prompts are then displayed (shown with sample responses):

```
Employee Information:
Enter 1 to Search Employee Information
Enter 2 to Enter Employee Information: >2

Enter Employee Last Name: >Smith

Enter Employee First Name: >Joseph

Enter Employee ID: >00001

Enter Employee office: >Raleigh

Enter Employee hire date (yyyy-mm-dd): >2005-01-01
```

The application then updates the database with the appropriate information. To query the database and view the information, run the application again, this time entering `1` to search employee information:

```
Employee Information:
Enter 1 to Search Employee Information
Enter 2 to Enter Employee Information: >1
```

At the prompt, enter the name of the employee described in the previous step to view the information in the database.

```
Enter Employee last name: >Smith

Last Name: Smith
FirstName: Joseph
Employee ID: 00001
Office: Raleigh
Hire Date: 2005-01-01 00:00:00.000
```

Summary

PERL is a highly portable language that provides a simple and flexible way to develop applications that are too intensive for shell commands, but do not need a complex language structure like C. For developers to deliver truly portable applications, DataDirect Connect *for* ODBC provides consistent implementations and performance across all major operating systems and databases. Because PERL applications are often used in Web-based scenarios, they need to be highly scalable and fast. DataDirect Connect *for* ODBC delivers both performance and scalability through the use of its unique wire protocol architecture. This architecture also does not require database client libraries for connecting to the database, easing deployment of the applications across environments. DataDirect Connect *for* ODBC is the fastest, most comprehensive suite of ODBC drives for all major databases.

We welcome your feedback! Please send any comments concerning documentation, including suggestions for other topics that you would like to see, to:

docgroup@datadirect.com

FOR MORE INFORMATION

800-876-3101

Worldwide Sales

Belgium (French).....0800 12 045
Belgium (Dutch).....0800 12 046
France.....0800 911 454
Germany 0800 181 78 76
Japan 0120.20.9613
Netherlands.....0800 022 0524
United Kingdom.....0800 169 19 07
United States.....800 876 3101

DataDirect Technologies is focused on data access, enabling software developers at both packaged software vendors and in corporate IT departments to create better applications faster. DataDirect Technologies offers the most comprehensive, proven line of data connectivity components available anywhere. Developers worldwide depend on DataDirect Technologies to connect their applications to an unparalleled range of data sources using standards-based interfaces such as ODBC, JDBC and ADO.NET, as well as cutting-edge XML query technologies. More than 250 leading independent software vendors and thousands of enterprises rely on DataDirect Technologies to simplify and streamline data connectivity. DataDirect Technologies is an operating company of Progress Software Corporation (Nasdaq: PRGS).

www.datadirect.com