



SQLBase Open Connectivity

By Suren Behari - Team Developer Product Manager
and Charles McLouth - SQLBase Product Manager

December, 2005

GUPTA™

Contents

Contents.....	2
Abstract	4
Introduction.....	4
SQLBase Interfaces	4
Middleware	5
Language Support for SQLBase Interfaces	7
Survey of APIs.....	7
SQL/API (SQL Application Programming Interface).....	8
What Is SQL/API?	8
Using SQL/API	8
SQL/API's suitability for various development environments.....	9
ODBC (Open Data Base Connectivity).....	10
What is ODBC?	10
SQLBase ODBC Driver	10
Using ODBC	10
Using PHP.....	12
ODBC's Suitability for Various Development Environments.....	13
JDBC (Java Data Base Connectivity)	13
What Is JDBC?.....	13
Using JDBC	13
JDBC's Suitability for Various Development Environments.....	14
OLE DB	14
What Is OLE DB?	14
Features of SQLBase OLE DB Data Provider	14
Using OLE DB.....	15

.NET Data Provider (NDP) and ADO.NET.....	15
What is ADO.NET?	15
What is a .NET Data Provider (NDP)?	16
Using SQLBase .NET Data Provider	16
SAL (Scalable Application Language)	17
What is SAL?.....	17
Using SAL	17
Pros and Cons of the APIs.....	18
Summary Table	19



Abstract

This paper explains the various SQLBase connectivity interfaces along with their respective advantages and differences. It will help you determine which interface is best suited for your application development efforts, based upon the development tool you are using and the needs of your application.

Introduction

GUPTA's SQLBase product line provides application developers with a full Relational Database Management System (RDBMS) designed for PC class hardware. Available in both server (workgroup) and local engine (standalone) configurations on Microsoft Windows and Linux, SQLBase delivers the power of Structured Query Language (SQL), transaction processing, and distributed processing for building business applications. By supporting the client/server architecture, SQLBase achieves a level of scalability that makes it well suited for a wide range of applications - from a standalone, single-user environment to a multi-user workgroup environment.

To build the client side of applications, developers are using various tools. Some of the more popular application development environments include Visual Basic, C/C++, Visual Studio .NET, PowerBuilder, Java, Delphi, PHP on Linux, and of course, Team Developer. Accessing SQLBase from an application development tool is accomplished using one of SQLBase's connectivity interfaces.

SQLBase Interfaces

A large variety of developer tools are used in the creation of the client-side of application systems, and database systems are no exception. The reasons for a large variety include the following (and others not listed):

- Historical precedent
- Skills are present in the local labor supply
- Dictated by corporate management

Whatever the reason, most development projects that require interfacing to a database cannot have a programming language thrust upon them. This means that access to the database engine must support a myriad of development tools.

Amongst the most popular application development environments for Windows (on Intel processors) client-server development are Visual Basic, C/C++, Visual Studio.NET, Power Builder, Java, Delphi, PHP for Linux, and GUPTA Team Developer.

Access to the database engine must support a myriad of database tools



Any database intending to be taken seriously as worthy for creating line-of-business or shrink-wrapped applications has got to support almost all of these.

Accessing SQLBase from an application development tool is accomplished using one of SQLBase's connectivity interfaces. Each of the SQLBase's Application Programming Interfaces (API) has its advantages and limitations.

Understanding these and when to use each of the interfaces is what this paper is all about.

**Connectivity
interfaces fall
into the category
of middleware**

Middleware

Connectivity interfaces fall into the category of middleware. Middleware provides the link for data exchange between the different points of processing in a distributed application. In a SQLBase application the points of processing are SQLBase itself and a client application.

Even in a standalone application environment, SQLBase and the client application are separate processes running on the same machine. The client/server architecture is maintained and a connectivity interface is required to support data exchange between the application and SQLBase.



SQLBase offers a choice of connectivity interfaces for linking client applications with SQLBase database

SQLBase Connectivity Interfaces

SQLBase offers a choice of connectivity interfaces for linking client applications with SQLBase databases. You can choose the connectivity interface that best suits your needs based on the development tool/language you are using and your application requirements:

- SQL/API is a low level, high performance interface suited for use with C and languages that support external functions in DLLs or shared objects. It provides complete access to the SQLBase feature set, including "administrative" operations.
- The SQLBase ODBC driver provides a simple and portable interface via ODBC. SQLBase fully supports multi-threaded ODBC applications on both Windows and Linux from such tools/languages as PHP.
- The SQLBase JDBC driver provides a platform independent native interface for Java applications and applets.
- The SQLBase OLE DB provider allows developers of Visual Studio and Delphi to write applications that use SQLBase.
- The SQLBase .Net Data Provider (NDP) provides a native interface for .Net applications using Visual Studio.Net (including C#, VB.NET, J#, and ASP.NET.)
- SAL (Scalable Application Language) is the fourth generation scripting language, or 4GL, used in GUPTA's Team Developer for coding business logic.

Choosing an interface does not mean that all of your application's access to SQLBase must be done exclusively through that one method. You can mix methods, but it should be done carefully and with an understanding of the issues it can bring up and the potential burden on the programmer who attempts it. When multiple interfaces are used, there is little synchronization of the requests and data used - the programmer must be the arbiter. Also in certain cases, care must be exercised to ensure database access is limited to one of the active interfaces to ensure integrity and reliability.

Language Support for SQLBase Interfaces

The grid below details the support for the SQLBase development APIs on a language-by-language basis.

A check mark indicates that the language-API pair is a good candidate for successful delivery. If a grid intersection is not checked, this does not mean that the combination cannot be made to work, only that it is not a recommended pairing.

Language	SQL/API	NDP	ODBC	JDBC	OLE DB
C (Windows or Linux)	✓		✓		✓
C++	✓		✓		✓
Visual Basic (6.0)	✓		✓		✓
Visual Studio.NET	✓	✓	✓		✓
Delphi	✓		✓		✓
Java				✓	
PowerBuilder	✓		✓		✓
Team Developer	✓				✓
PHP (Windows or Linux)			✓		

Survey of APIs

We will now discuss the following APIs:

- SQL/API (SQL Application Programming Interface)
- ODBC (Open Data Base Connectivity) API
- JDBC (Java Data Base Connectivity)
- OLE DB (ADO)
- NDP (ADO.NET)
- SAL (Scalable Application Language)



SQL/API is Gupta Technologies' proprietary SQLBase RDBMS API

SQL/API (SQL Application Programming Interface)

What Is SQL/API?

This is Gupta Technologies' proprietary SQLBase RDBMS API, which exposes the database engine's complete functionality. The function set provides SQL access to SQLBase data as well as non-SQL specific database and server management operations.

The Structured Query Language/Application Programming Interface (SQL/API) is a function library designed for use with the C programming language on Windows or Linux, and development environments that support C-style external function calling conventions. SQL/API is a call level interface (CLI) analogous to SQL*NET in Oracle environments and CT-LIB in Sybase® environments.

Typical function calls include connecting and disconnecting to a database, passing SQL statements to the server for compilation and execution, providing bind variable data, and retrieving result sets. SQL/API provides functions to perform administrative tasks such as performing database backups and restorations.

Using SQL/API

To interact with SQLBase, you make calls to SQL/API functions throughout your application.

The following C code provides an example of typical SQL/API usage for database applications:

```

SQLTCUR cur; /* Cursor Handle */
SQLTRCD rcd; /* Return Code */

sqlini((SQLTPFP) (0));
if (rcd = sqlenc( &cur, "SALES/MANAGER/A1S2D3F4" )
{
    printf("connection failure (rcd = %d)\n", rcd);
    exit(0);
}
else
{
    printf("connection established\n");
    static char sqlcmd[] =
"SELECT NAME, DEPT FROM SALES_TEAM WHERE ID = :1";
    char id[] = "FRED";
    #DEFINE COLWIDTH 30
    char dataline[80];
    unsigned char cvl;
    char fcd;
    char *lp = dataline;
    SQLTSLC col;
    rcd = sqlcom(cur, sqlcmd, 0 );
    rcd = sqlbnn(cur, 1, id, sizeof(id), 0, SQLPBUF);
    rcd = sqlexe(cur);
    memset( dataline, '-', sizeof(dataline));
    for (col=1, col<=2, col++)
    {

```



```

        ret = sqlssb(cur, col, SQLPBUF, lp, COLWIDTH, 0, &lc, &fcd);
        lp += (COLWIDTH + 2);
    }
    while(!rcd = sqlfet(cur))
    {
        println("%s", dataline)
    };
    sqldis(cur);
}

```

A simple trace through the code: After the library is initialized with `sqlini`, `sqlcnc` is used to connect a cursor to the database. Then `sqlcom` is used to send a SQL statement to the database server for compilation. `sqlbnn` is used to bind a value with the bind variable in the SQL statement, after which the compiled SQL statement can be executed using `sqlexe`. To prepare for retrieving data, `sqlssb` calls are made to setup a buffer for each item in the SELECT statement. This is followed by fetching the data from the result set on the server to the client application using `sqlfet`. Finally, the cursor is disconnected using `sqldis`.

SQL/API's suitability for various development environments

The SQL/API provides complete access to the SQLBase feature set, and includes functions to perform administrative tasks like creating new databases, and functions for performing database backups and restorations. If you need full access to the complete feature set, then the SQL/API is an appropriate interface to use.

Languages that are based on the same intrinsic data types as the C-language present the least effort in implementing SQL/API. As the level of isolation from the computer hardware raises the work required to build interfaces to call SQL/API increases and the results suffer.

The ability to control DBA functions programmatically is particularly useful for applications that are deployed in a mobile or isolated environment because it allows database maintenance activities to be performed with no user intervention.

When using the SQL/API directly, you need to understand the context around its design. Originally designed for use with the C language, it is packaged as a set of header files and object libraries for C development environments. The documentation is written with C programmers in mind. The sample programs are in C, and so on.

This does not mean SQL/API cannot be used with other programming languages and development environments; all of the common development environments such as Team Developer, Visual Basic, Delphi, Power Builder, etc. provide sufficient support to exploit SQL/API effectively.

However the *impedance mismatch* (the difference between the design intentions of the programming language and the API used) that can develop with some high-level 4GLs may be problematic. The most common problem being that developers who work in these languages are often not used to dealing with the level of detail required in a lower level interface like the SQL/API.



ODBC (Open Data Base Connectivity)

What is ODBC?

ODBC, as presented by the SQL Access Group in 1990 and by Microsoft in 1991, has been the standard adopted by virtually all data-oriented desktop tool suppliers.

Open Database Connectivity (ODBC) provides an open, vendor independent interface to database systems for applications. Based on the X/Open CLI specification, Microsoft has promoted ODBC to the point where it is now a widely accepted standard. ODBC is largely language independent though it was originally designed for C environments, as was the SQL/API. ODBC 3.0 fully implements the X/Open CLI specifications and adds features commonly needed by developers of screen-based database applications, such as scrollable cursors. ODBC is also available on Linux using the iODBC Driver Manager or the unixODBC Driver Manager.

The SQLBase ODBC driver is implemented as a layer on top of the SQL/API. This driver allows developers to concentrate on writing business logic with the development tool of their choice, without having to implement the specifics and complexities of their middleware API. For programmers using tools like Visual Basic, Power Builder, Delphi, C, PHP, or other popular based development environments for Windows or Linux; ODBC is the easiest and quickest way to develop an application.

SQLBase ODBC Driver

GUPTA provides an ODBC interface to SQLBase through a driver developed in-house that conforms to the ODBC 3.5 specification. The ODBC driver for SQLBase version 7.6.1 was developed by Merant (now DataDirect Technologies) and conforms to the Microsoft ODBC 3.0 specification. The new driver supports a consistent level of ODBC Core, Level 1 and Level 2 functions, and is backward compatible with the previous version from Merant. Newer versions of GUPTA SQLBase also include a Linux ODBC driver that supports both the iODBC Driver Manager and unixODBC Driver Manager.

The ODBC driver is multi-threaded which allows for greater scalability of ODBC applications. By supporting most of the advanced 3.x ODBC features like bookmarks, descriptors and diagnostic APIs, the driver for SQLBase allows one to better integrate SQLBase with Microsoft Visual Studio and PHP. The ODBC driver provides support for the deprecated 2.x driver functions which is transparent to the application, thus allowing existing applications to work with the new driver against any version of SQLBase.

Using ODBC

For a comparison, here is the same example in ODBC form:

```
Visual Basic using Remote Data Objects
Sample form
Sample Code
Option Explicit
Private WithEvents envSales As rdoEnvironment
```

The ODBC driver for SQLBase supports Linux & Windows

```

Private WithEvents c onSales As rdoConnection
Private rstSales As rdoResultset
'-----
Private Sub ShowRow()
    txtSalesDat(1).Text = rstSales!Name
    txtSalesDat(2).Text = rstSales!Dept
End Sub
'-----
Private Sub cmdClose_Click()
    Unload Me
End Sub
'-----
Private Sub cmdFind_Click()
    Dim strSQL As String
    On Error GoTo CheckError
    strSQL = "SELECT Name, Dept FROM Sales_Team WHERE Id = "
        & txtSalesDat(0)
    Set rstSales = conSales.OpenResultset(strSQL, rdOpenStatic,
        rdConcurReadOnly)
    rstSales.MoveFirst
    ShowRow
SubExit:
Exit Sub

CheckError:
    Dim err As rdoError
    Dim strMsg As String
    For Each err In rdoErrors
        strMsg = strMsg & er & vbCrLf
    Next
    MsgBox "Error creating result set " & strMsg
    Resume SubExit
End Sub
'-----
Private Sub cmdNavigate_Click(Index As Integer)
    Select Case Index
    Case 0 ' Move to previous row
        rstSales.MovePrevious
        If rstSales.BOF Then
            Beep
            rstSales.MoveFirst
        End If
    Case 1 ' Move to next row
        rstSales.MoveNext
        If rstSales.EOF Then
            Beep
            rstSales.MoveLast
        End If
    End Select
    ShowRow
End Sub
'-----
Private Sub Form_Load()
    On Error GoTo CheckError
    Set envSales = rdoEngine.rdoCreateEnvironment("Sales", "", "")
    Set conSales = envSales.OpenConnection("Sales",
        rdDriverNoPrompt, True, "DSN=Sales;UID=" &
        strUserID & ";PWD=" & strPWD & ";Database=NASales")

```

```

SubExit:
Exit Sub

CheckError:
Dim err As rdoError
Dim strMsg As String
For Each err In rdoErrors
    strMsg = strMsg & er & vbCrLf
Next
MsgBox "Could not create environment & connection. " & strMsg
Resume SubExit
End Sub
'-----
Private Sub Form_Unload(Cancel As Integer)
'Checking to ensure user did connect; otherwise close will generate an
'error

    If rstSales.ActiveConnection = "Sales" Then
        rstSales.Close
        conSales.Close
        envSales.Close
    End If
End Sub

```

Using PHP

Here is a PHP sample using the Unified ODBC Functions:

```

<!--phpsample.php-->
<html>
<body>
<?php
    $conn=odbc_connect('guptadsn','SYSADM','SYSADM');
    if (!$conn)
    {
        exit("Connection Failed: " . $conn);
    }
    $sql="SELECT * FROM COMPANY";
    $rs=odbc_exec($conn,$sql);
    if (!$rs)
    {
        exit("Error in SQL");
    }
    echo "<table><tr>";
    echo "<th>Company_ID</th>";
    echo "<th>Company_Name</th></tr>";
    while (odbc_fetch_row($rs))
    {
        $compid=odbc_result($rs,"COMPANY_ID");
        $compname=odbc_result($rs,"COMPANY_NAME");
        echo "<tr><td>$compid</td>";
        echo "<td>$compname</td></tr>";
    }
    odbc_close($conn);echo "</table>";
?>
</body>
</html>

```



ODBC's Suitability for Various Development Environments

As a standardized, open and independent interface, ODBC is well suited for use with development environments that do not have a customized interface to SQLBase. This would include tools like Visual Basic, Power Builder, Delphi, and PHP.

ODBC is also suitable for use with C. A compelling reason would be for a C programmer, experienced with ODBC, is just starting to develop applications with SQLBase and needs an optimally productive interface.

Some development environments add an additional layer of ease-of-use on top of ODBC, such as Visual Basic's ADO, Visual Studio.NET's ADO.NET, and PHP's Unified ODBC Functions.

JDBC (Java Data Base Connectivity)

What Is JDBC?

The SQLBase JDBC Driver is a set of Java classes (and interfaces) that provide a standard API for connecting to and accessing SQL databases. The SQLBase JDBC driver is a level 4 driver, which means it's a pure Java driver that communicates with SQLBase using a low level protocol known as the Message Level Interface (MLI). No other gateways or other layers of middleware are needed. Hence, the SQLBase JDBC driver is well suited for Java applets on the Web and company Intranets.

Using this driver, developers creating Java applets or full Java applications on any platform that supports the Java Virtual Machine (JVM) can connect to SQLBase.

For reference, the SQLBase JDBC driver supports JDK 1.1.x standard and JDK1.2 (only limited).

SQLBase has a JDBC driver that provides an interface to SQLBase from Java applications and applets. This driver implements the object-oriented interface defined by JavaSoft for JDBC drivers. GUPTA has registered the sub protocol name *sqlbase* for use with the SQLBase JDBC driver.

Using JDBC

Here is a JDBC version of the previous example:

```

try
{
    Class.forName("jdbc.gupta.sqlbase.SqlbaseDriver")
    String URL = "jdbc:sqlbase://localhost/sales";
    String query = "SELECT NAME, DEPT FROM SALES_TEAM
                  WHERE ID = ? ";
    Connection con =
    DriverManager.getConnection( URL, "MANAGER", "A1S2D3F4" );
    PreparedStatement stmt = con.prepareStatement(query);
    stmt.setString(1, "FRED");
    String name, dept;
    ResultSet rs = stmt.executeQuery( );

```



```

While(rs.next())
{
name = rs.getString(1);
dept = rs.getString(2);
}
}
catch(SQLException ex)
{
// do error handling
}

```

Again, notice the elegance and simplicity provided by an object oriented approach.

JDBC's Suitability for Various Development Environments

The SQLBase JDBC driver is suitable for use in building Java applications and applets. It is not callable from other language environments. It will work with the raw JDK and all of the popular Java IDEs.

OLE DB

What Is OLE DB?

GUPTA's SQLBase OLE DB Data Provider is an important part of Microsoft's Universal Data Access strategy. The strategy provides high-performance access to all types of information from any data source on platforms ranging from desktops to enterprise servers. For example developers using Visual Studio or Visual Studio.NET can use OLE DB to create high performance database applications using SQLBase.

It is a set of COM (Component Object Model) interfaces that provides applications with uniform access to data stored in diverse information sources. These interfaces support the amount of DBMS functionality appropriate to the data source, enabling it to share its data. OLE DB data providers also make high-level access to data, through ActiveX® Data Objects (ADO and ADO.NET), possible.

GUPTA's SQLBase OLE DB Data Provider takes advantage of this strategy to simplify access to SQLBase functionality from third-party programming tools such as Visual Basic, Delphi, and C++. Since the OLE DB Provider provides a set of COM interfaces, consumers can access information from SQLBase from any language.

As with other OLE DB data providers, the COM object will be developed as a Dynamic Link Library (DLL). In addition to standard OLE DB conformance, interfaces are provided with the SQLBase OLE DB Data Provider that enable access to DBMS functionality not normally provided by OLE DB, such as backup, create database, shutdown server, etc.

Features of SQLBase OLE DB Data Provider

Some of the many advantages of using SQLBase OLE DB Data Provider are:

- The ability to access SQLBase data through any language that supports COM, such as C++, Visual Basic and Delphi. It allows

**SQLBase OLE DB
Provider supports
COM+
transactions**

different programmers in an organization to access the same data in the same way; regardless of what language they use.

- The ability to expose SQLBase data to other data sources such as SQL Server, Microsoft Excel and Access. This can be very useful for transfer data amongst different formats.
- Additional COM objects included that allow access to non-OLE DB functionality of the server, such as *backup/restore*, *create/delete database* and *shutdown server*.
- Encryption of data on the wire when used with SQLBase TE (Treasury Edition). COM+ transactions allow spanning transactions over databases on multiple locations and from multiple providers. This advanced enterprise integration feature is available in the SQLBase OLE DB provider as well.

Using OLE DB


Here is an OLE DB version of the same example used earlier:

```
Public oAdoConn As New ADODB.Connection
Public oAdoRS As New ADODB.Recordset
Public oAdoCMD As New ADODB.Command
'
' Processing for the Find button
Private Sub cmdFind_Click()
Private Sub cmdFind_Click()
    Dim sStatement, sID As String
    Dim param1 As ADODB.Parameter
    Call oAdoConn.Open("Provider=SQLBASEOLEDB;UserID=" & _
    "MANAGER, Password=A1S2D3F4, Persist Security" & _
    "Info=False")
    sStatement = "SELECT NAME, DEPT FROM SALES_TEAM " & _
    "WHERE ID = :sID "
    oAdoCMD.CommandText = sStatement
    Set param1 = oAdoCMD.CreateParameter("sID", adVarChar, _
    adParamInput, 50)
    oAdoCMD.Parameters.Append param1
    param1.Value = "FRED"
    Set oAdoCMD.ActiveConnection = oAdoConn
    Set oAdoRS = oAdoCMD.Execute
    oAdoRS.MoveFirst
    Call MsgBox(oAdoRS(0) & "-" & oAdoRS(1), vbInformation + _
    vbOKOnly, "Message")
SubExit:
    Exit Sub
End Sub
```

.NET Data Provider (NDP) and ADO.NET

What is ADO.NET?

ADO.NET is a set of classes that expose data access services to the .NET programmer. ADO.NET provides a rich set of components for creating distributed, data-sharing applications. It is an integral part of the .NET Framework, providing access to relational data, XML, and



application data. ADO.NET supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects used by applications, tools, languages, or Internet browsers.

ADO.NET provides consistent access to data sources such as SQLBase, as well as data sources exposed through OLE DB and XML. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update data.

ADO.NET cleanly factors data access from data manipulation into discrete components that can be used separately or in tandem. ADO.NET includes .NET Framework data providers (also known as .NET Data Provider or NDP) for connecting to a database, executing commands, and retrieving results. Those results are either processed directly, or placed in an ADO.NET DataSet object in order to be exposed to the user in an ad-hoc manner, combined with data from multiple sources, or remote between tiers. The ADO.NET DataSet object can also be used independently of a .NET Framework data provider to manage data local to the application or sourced from XML.

What is a .NET Data Provider (NDP)?

A .NET data provider is used for connecting to a database, executing commands, and retrieving results. Those results are either processed directly, or placed in an ADO.NET DataSet in order to be exposed to the user in an ad-hoc manner, combined with data from multiple sources, or remote between tiers. The .NET Framework data provider is designed to be lightweight, creating a minimal layer between the data source and your code, increasing performance without sacrificing functionality.

Using SQLBase .NET Data Provider

Example:

```
' Declare variables for the Connection, Adapter & DataSet
Dim conn As New SQLBaseConnection()
Dim sStatement, sID, sName, sDept, sTitle
Dim cmd As New SQLBaseCommand
Dim dr As SQLBaseDataReader
'

' Handle the Click event for the Find button
Public Sub cmdFind_Click(ByVal sender As Object, ByVal e As _
    System.EventArgs) Handles cmdFind.Click
    conn = New SQLBaseConnection("data source=island; uid "& _
        "= sysadm; pwd= sysadm; ini=D:\Gupta\SQLBase" & _
        " 8.5\sql.ini")
    conn.Open()
    sStatement = "SELECT CITY, COMPANY_NAME FROM "& _
        "COMPANY WHERE CITY = :sID"
    cmd = New SQLBaseCommand(sStatement, conn)
    cmd.Parameters.Add("sID", DbType.String).Value = _
        "San Francisco"
    dr = cmd.ExecuteReader()
    While (dr.Read())
        MessageBox.Show(dr.GetString(0) + ", " +
            dr.GetString(1), sTitle, MessageBoxButtons.OK)
```

```
End While
dr.Close()
conn.Close()
End Sub
```

SAL (Scalable Application Language)

What is SAL?

SAL provides an extensive family of functions that build upon the SQL/API

Scalable Application Language (SAL) is the object oriented 4GL scripting language of GUPTA's development tool Team Developer. SAL provides an extensive family of functions that build upon the SQL/API to provide developers with a high level interface to SQLBase.

SAL's basic constructs are similar to those of other "standard" languages (*if, while, case, etc.*), but the language is heavily centered on the development of database applications. Contributing strongly to this affinity to database applications is the fact that the interface to SQLBase is embedded in the SAL language itself.

As a blend of 4GL and API, SAL removes most of the complexities of the SQL/API, such as setting up buffers to receive the fetched data from a query result set. SAL also integrates the language constructs with the database interface so that graphical widgets such as text fields can be used as bind variables with SQL statements.

SAL is designed for writing the client side business logic and painting the user interfaces for client/server applications. This design intent means that the database functions that SAL provides purposely do not include functions for administrative work such as creating databases or performing backups.

This apparent limitation in SAL is not a handicap since SAL has the ability to call external functions in DLLs and it is possible to use the SQL/API directly from SAL. Since SAL itself incorporates a robust high level interface to SQLBase, it is very compelling to just use SAL for the vast majority of application tasks and resort to SQL/API only when needed. The cases that come to mind for requiring SQL/API are applications that programmatically manipulate the database engine itself.

Using SAL

Here is a SAL equivalent of the earlier example presented for the SQL/API.

Variables

```
Sql Handle: hSql
String: sID
String: sStatement
String: sName
String: sDept
```

Actions

```
Set SqlDatabase = "SALES"
Set SqlUser = "MANAGER"
Set SqlPassword = "A1S2D3F4"
When SqlError
    ! Error handling logic would go here
```

```

Call SqlConnection( hSql )
Set sStatement = "SELECT NAME, DEPT FROM SALES_TEAM
                WHERE ID = :sID INTO :sName, :sDept"
Set sID = "FRED"
Call SqlPrepare( hSql, sStatement)
Call SqlExecute( hSql )
While SqlFetchNext( hSql, nFetchInd )
    Call SalMessageBox( sName || "-" || sDept, sTitle,
                      MB_Ok )
Call SqlDisconnect( hSql )

```

Notice how much easier it is to read and follow the SAL example, the benefit of using a higher-level interface.

SAL is the scripting language for SQLWindows and GUPTA's Team Developer and its integrated SQLBase connectivity features make it the obvious choice to use with these environments.

Pros and Cons of the APIs

The following table is a digested view of the content of the previous interface reviews, presented in a format that enables comparison.

API	Pro	Con
SQL/API	<ul style="list-style-type: none"> Best possible performance Advanced functionality. Windows or Linux. 	<ul style="list-style-type: none"> Developer must take care of everything. Code is less portable, cannot mix databases.
ODBC	<ul style="list-style-type: none"> Highly portable code. Industry standard. Almost all tools will support it. Windows or Linux. 	<ul style="list-style-type: none"> Database operations against SQLBase are limited to database operations (no server control). Limited support for advanced features of product.
OLE DB	<ul style="list-style-type: none"> COM+ Transaction Support. Support for Microsoft's Universal Data Access. strategy Ease of use through abundant higher level object models. 	<ul style="list-style-type: none"> Added overhead on performance (depends on tool object model). Limited support for advanced features of product. Windows only.
JDBC	<ul style="list-style-type: none"> 100% pure Java. No overhead from bridge. Platform independent. 	<ul style="list-style-type: none"> Java only.
NDP	<ul style="list-style-type: none"> Tight integration with Microsoft's .NET initiative. Ease of use through ADO.NET Object Model. 	<ul style="list-style-type: none"> .NET only Limited support for advanced features of product. Added overhead on performance (depends on tool object model). Windows only.
SAL	<ul style="list-style-type: none"> SQLBase and SAL are tightly integrated. Optional integration of both SAL and SQL/API interfaces. 	<ul style="list-style-type: none"> Team Developer only.

Summary Table

Interface	C	C++	Team Developer	Visual Basic	Visual Studio.NET	Delphi	Java	PHP
SQL/API	A	B	B	B	B	B	N/A	N/A
ODBC	A	B	N/A	A	B	A	N/A	A
JDBC	N/A	N/A	N/A	N/A	N/A	N/A	A	N/A
OLE DB	B	A	A	A	A	A	N/A	N/A
NDP	N/A	N/A	N/A	N/A	A	N/A	N/A	N/A

Legend:

A – Interface is well suited for use with development environment

B – Interface works sufficiently well with development environment

C – Interface will work with development environment

N/A – Interface is not appropriate for use with development environment