instruction manual

i!-DatabasePlus and DBWizard

integration!Solutions

integration!Solutions

Software Warranty Agreement

GRANT OF LICENSE. AMX Corporation grants to you the right to use the enclosed **i!-DatabasePlus and DBWizard** software program (the SOFTWARE) on a single central processing unit (CPU). This license is for object code only and does not convey any rights to use of the source code.

This Agreement does not authorize you to distribute the SOFTWARE.

COPYRIGHT. The SOFTWARE is owned by AMX Corporation, and is protected by United States copyright laws and international treaty provisions. Therefore, you must treat the SOFTWARE like any other copyrighted material (e.g., a book or musical recording) except that you may either (a) make one copy of the SOFTWARE solely for backup or archival purposes, or (b) transfer the SOFTWARE to a single hard disk provided you keep the original solely for backup or archival purposes. You may not copy the written materials accompanying the SOFTWARE.

OTHER RESTRICTIONS. You may not rent or lease the SOFTWARE, but you may transfer the SOFTWARE and accompanying written materials on a permanent basis provided you retain no copies and the recipient agrees to the terms of this Agreement. You may not reverse engineer, decompile, or disassemble the SOFTWARE. If the SOFTWARE is an update or has been updated, any transfer must include the most recent update and all prior versions.

You may use only one version of the SOFTWARE at any time. You may not use the version of the SOFTWARE not being run on your CPU on any other CPU or loan, rent, lease or transfer them to another user whether with or without consideration.

LIMITED WARRANTY

LIMITED WARRANTY. AMX Corporation warrants that the SOFTWARE will perform substantially in accordance with the accompanying written materials for a period of ninety (90) days from the date of receipt. Any implied warranties on the SOFTWARE and hardware are limited to ninety (90) days and one (1) year, respectively. Some states/countries do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you.

CUSTOMER REMEDIES. AMX Corporation's entire liability and your exclusive remedy shall be, at AMX Corporation's option, either (a) return of the price paid, or (b) repair or replacement of the SOFTWARE that does not meet AMX Corporation's Limited Warranty and which is returned to AMX Corporation. This Limited Warranty is void if failure of the SOFTWARE or hardware has resulted from accident, abuse, or misapplication. Any replacement SOFTWARE will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

NO OTHER WARRANTIES. AMX Corporation disclaims all other warranties, either expressed or implied, including, but not limited to implied warranties of merchantability and fitness for a particular purpose, with regard to the SOFTWARE, the accompanying written materials, and any accompanying hardware. This limited warranty gives you specific legal rights. You may have others which vary from state/country to state/country.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES. In no event shall AMX Corporation be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use of or inability to use this AMX Corporation product, even if AMX Corporation has been advised of the possibility of such damages. Because some states/countries do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

U.S. GOVERNMENT RESTRICTED RIGHTS

The SOFTWARE and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software--Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is AMX Corporation, 3000 Research Drive, Richardson, TX 75082.

If you acquired this product in the United States, this Agreement is governed by the laws of the State of Texas.

Should you have any questions concerning this Agreement, or if you desire to contact AMX for any reason, please write: ***AMX Corporation, 3000 Research Drive, Richardson, TX 75082***.

Table of Contents

Introduction

i!-DatabasePlus

i!-Database Plus™ is an application that allows you to connect a NetLinx™ Master to a server or PC database. The kit consists of two files:

- **NetLinxDBInclude.asp**: A server database script designed to run on Microsoft web servers.
- **NetLinxDBInclude.axi**: Includes various functions to help you integrate to a server database script.

The two files work together to allow NetLinx to access any ODBC database that the server can connect to.

These two files connect to each other using all of the following:

- Hypertext Transfer Protocol (HTTP): the base protocol to which CGI and XML are then added.
- Common Gateway Interface (CGI): used by the NetlinxDBInclude.axi file to make database requests to the NetlinxDBInclude.asp file.
- Extensible Markup Language (XML)

DBWizard

DBWizard™ is an application that helps you generate NetLinx code to access a PC database. The NetLinx Master can read and write to the database via a web server connection. DBWizard helps generate the SQL and NetLinx code needed to read and write to the database.

To access a database using NetLinx, you will need the following:

- A PC running a Microsoft Window© operating system such as Windows 95®, Window 98®, Windows NT 4.0® Workstation, Windows NT Server®, Windows 2000® Workstation or Windows 2000® Server.
- A Web server. You can use Microsoft Internet Information Server (IIS) or Microsoft Personal Web Server (PWS); either is available for all operating systems listed above.
- A NetLinx Master with Ethernet

Supported Operating Systems

- Windows 95/98 (with at least 48 MB of installed memory)
- Windows NT 4.0 Workstation or Server (service pack 6 B or greater, with at least 64 MB of installed memory)
- Windows 2000 Professional or Server (running on a Pentium 233 MHz processor (minimum requirement); 300 MHz or faster recommended, with 96 MB of installed memory.)

Minimum PC Requirements

- Windows-compatible mouse (or other pointing device)
- At least 5 MB of free disk space (150 MB recommended)
- VGA monitor, with a minimum screen resolution of 800 x 600
- A Network adapter
- A Web server such as Personal Web Server (PWS) or Internet Information Server (IIS).
  - Windows 95®/98®, and NT 4.0® Workstation uses PWS.
  - Windows 2000® Professional or Server, and Windows NT 4.0® Server uses IIS.

Installing i!-Database Plus

1. In Explorer, double-click **i!-DatabasePlusSetup.exe** from the directory window where you downloaded the i!-Database Plus install program.
2. After reading the License Agreement, select **I Agree** and **Next** to proceed.
3. The Welcome To i!-Database Plus Setup dialog appears, reminding you to close all Windows programs before going any further. Click **Next** to proceed.
4. In the i!-DatabasePlus Select Components dialog, select which example programs you would like to install.
5. In the Select i!-Database Plus Install Location dialog, use the **Browse** button to navigate to a directory other than the default install directory, if desired. Click **Next**.
6. In the i!-Database Plus Shortcut Creation dialog, select Install Shortcut Icons for the installed components on your desktop, if desired.
7. Click **Next** in the Start i!-Database Plus Installation dialog to install the selected components.
8. The program prompts you to restart your system to complete the installation.

Installing DBWizard

1. In Explorer, double-click **DBWizardSetup.exe** from the directory window where you downloaded the DBWizard install program.
2. After reading the License Agreement, select **I Agree** and **Next** to proceed.
3. The Welcome To DBWizard Setup dialog appears, reminding you to close all Windows programs before going any further. Click **Next** to proceed.
4. In the Select DBWizard Install Location dialog, use the **Browse** button to navigate to a directory other than the default install directory, if desired. Click **Next**.
5. In the DBWizard Shortcut Creation dialog, select **Install Shortcut Icons** for the installed components on your desktop, if desired.
6. Click **Next** in the Start DBWizard Installation dialog to install the selected components.
7. The program prompts you to restart your system to complete the installation.

Setup

NetLinxDBInclude.asp and NetLinxDBInclude.axi work in conjunction with your code to allow access to a PC database. In order to use these two files, you will need the following:

- A PC running a Microsoft Windows© operating system such as Windows 95, Window 98, Windows NT Workstation, Windows NT Server, Windows 2000 Workstation or Windows 2000 Server.
- A Web server. You can use Microsoft Internet Information Server (IIS) or Microsoft Personal Web Server (PWS). Either is available for all operating systems listed above.
- A NetLinx Master with Ethernet

You must have IIS or PWS installed and running ASP (Active Server Pages) before beginning. Make sure there were no errors during the installation of the web server, and you can call up an ASP file (any file ending in .ASP). If you get a **500 Internal Server Error** message when trying to view ASP pages in a web browser, the web server is not installed properly. You must correct this problem before proceeding.

Once the server PC is setup, make sure both the server PC and the NetLinx Master are connected to a network and can communicate with each other. A simple way to test connectivity is by using the PING program from the server PC's command line. Type **Ping <ip address of the NetLinx Master>** and you should see something like the following:

```
C:\WINDOWS\Desktop>ping 192.168.13.33
Pinging 192.168.13.33 with 32 bytes of data:
Reply from 192.168.13.33: bytes=32 time<10ms TTL=62
Reply from 192.168.13.33: bytes=32 time<10ms TTL=62
Reply from 192.168.13.33: bytes=32 time<10ms TTL=62
Reply from 192.168.13.33: bytes=32 time=1ms TTL=62
Ping statistics for 192.168.13.33: Packets:
    Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms

C:\WINDOWS\Desktop>
```

Now that your hardware is configured, you are ready to begin programming.

Programming i!-Database Plus

i!-Database Plus consists of creating three files:

- ASP file
- AXI file
- AXS file

If you already used DBWizard to generate an include file, you can skip to *Creating an AXS file* section on page 15 to finish your programming.

Prior to explaining how to program these files, a brief overview is necessary.

Data Source Name

Setting up a DSN (Data Source Name) is an easy way to provide a path to a database. It is also easy to change the path to the database or even the type of database without changing any code in the applications that use it. To setup a DSN:

1. Select **Start > Settings > Control Panel > ODBC Data Sources**, and select **System DSN** from the tabs at the top. Then, click **Add** to add a database.
2. Select the type of database to connect to. You will be given as many choices as you have ODBC drivers loaded. The screen allows you to enter the data source name. Pick something relating to the database you are connecting to.
   To connect to a database type not listed, contact the vendor and obtain the ODBC drivers for that database.
3. Connect to the database. This will vary depending on the database you chose. If you chose a Microsoft Access (MDB) database, click **Select** under database, and browse to the MDB file. When you are finished, click **OK** and the DSN will be added.

If you have chosen another type of database and are unsure about how to connect to the database, contact your database administrator.

Structure Query Language (SQL)

Structure Query Language, or SQL, is a common database language used to manipulate data in many different types of databases. In the process of writing the AXS file, you will need to generate SQL statements to access the database.

If you are not familiar with SQL, you have a couple of options:

- Learn SQL, or
- Use Microsoft Access to generate SQL statements for you.

You can use DBWizard™ to help generate the SQL statements and the NetLinx code. The DBWizard option is the easiest and fastest way to integrate to the database.

If you want to learn SQL, there are many resources available. You can find books at your local bookstore or use web sites. If you like reading books, check out "Lan Times, Guide to SQL" by

James R. Groff and Paul N. Weinberg. If you like to read on the web, surf to: http://www.informix.co.za/answers/english/docs/visionary/infoshelf/sqls/sqls.ix.html.

Another option is to read the Microsoft Access® help file. Microsoft has put a lot of good information regarding SQL into the help file.

If you are familiar with Access, build the queries in Access and switch to the SQL view from the View menu to see what you built. This provides a simple and intuitive way to build your SQL.

The last and easiest option is to use AMX's DBWizard program. This program peeks into your database and helps you build the SQL to read and write to any of the tables or views in the database. It also generates a large amount of the NetLinx code for you, so your development time is greatly reduced. This program is available from www.amx.com.

HTTP, CGI and XML

HTTP (Hyper-Text Transfer Protocol)

An example URL to retrieve records 1 - 5 from a table called Titles is:

http://server/databasescript.asp?SQL=SELECT * FROM Titles&Ps=5&start=21

where:

server	Name of web server
databasescript.asp	Name of server script
?SQL=SELECT * FROM Titles&Ps=5&start=21	CGI Parameters

You can use a URL like this to run a query directly against the server script, like the NetLinx system. This is often useful during testing to make sure the server script is operating properly.

*If you do use a URL like this, you will notice that once you hit enter, the spaces are replaced with **%20**. URL's have a certain set of characters that are excluded from normal use. These include space, question mark, ampersand, slash and colon. The reason for this is these characters are used in other parts of URL's. To use them again would cause confusion during URL parsing. One of the functions in the NetLinxDBInclude.axi file takes care of this conversion for you.*

CGI (Common Gateway Interface)

CGI provides a simple way to pass parameters to a program from an HTTP request. Most commonly, the CGI parameters appear directly on the URL line. For example: **http://www.somehost.com/index.htm?myname=amx&myhometown=dallas**. The first part of the URL looks familiar: it contains a protocol, a host name and a file to retrieve (protocol = **http://**, host=**www.somehost.com**, file=**/index.htm**). The last part is CGI. CGI parameters are passed in name-value pairs separated by an "**=**". The "**?**" is used to signify the start of CGI parameters. The "**&**" is used to separate CGI parameters. So this URL contains two CGI parameters: the first is a CGI parameter called **myname** and contains a value of **amx**. The second is **myhometown** and contains a value of **dallas**. The CGI application on the web server has access to these variables and can use them to help understand the context in which the request was made.

The NetLinx Database Gateway uses CGI to pass the database request to the server script. Both the NetLinxDBInclude.axi and NetLinxDBInclude.asp must agree to use the same CGI parameters to pass this information. The standard set of CGI parameters they use are in the following table.

CGI Parameters

Parameter	Parameter Name	Description	Notes
sql	SQL statement	The SQL statement passed to the database.	Required.
ps	Page Size	Optional. Number of records to retrieve from a record set.	Default is 10.
start	Start Index	Optional. Starting index in record set to retrieve values from.	Default is 1.
hdr	Header	Optional. Echoed by the server script to identify the XML packet.	Default is "Unknown".
abs	Absolute	Optional. Return indexes are absolute.	Default is relative.
pnl	Panel Index	Optional. Echoed by the server script to identify as Panel Index.	

XML (Extensible Markup Language)

Once the request is made, the server script opens the database, executes the SQL statement, and retrieves the results. Once the script has these results, it converts all the records into XML format. XML allows any data to be formatted in a structured way. XML is ideal for packaging up the results from the database query and returning them to the NetLinx Master. The NetLinx code can then parse this XML and extract the information the XML contains.

The XML format used by the NetLinx Database Gateway conforms to a certain standard so that the NetLinxDBInclude.axi always knows how find and parse the data. An example of this format is shown below (the actual data is highlighted to make it easier to read):

```
<rsHeader>
  <struct>
    <var><name>Start</name><data>1</data></var>
    <var><name>PageSize</name><data>10</data></var>
    <var><name>NumberRecords</name><data>3</data></var>
    <var><name>TotalRecords</name><data>3</data></var>
  </struct>
  <array>
    <struct>
      <index>1</index>
      <var><name>TitleID</name><data>11101000</data></var>
      <array><name>Artist</name><string>Buffet, Jimmy</string></array>
      <array><name>Title</name><string><![CDATA[Living & Dying in 3/4 Time]]>
      </string></array>
      <array><name>Copyright</name><string>MCA</string></array>
      <array><name>Label</name><string>MCA</string></array>
      <array><name>ReleaseDate</name><string>1974</string></array>
      <var><name>NumTracks</name><data>11</data></var>
    </struct>
    <struct>
      <index>2</index>
      <var><name>TitleID</name><data>17248229</data></var>
      <array><name>Artist</name><string>Buffet, Jimmy</string></array>
      <array><name>Title</name><string>Off to See the Lizard</string></array>
      <array><name>Copyright</name><string>MCA</string></array>
      <array><name>Label</name><string>MCA</string></array>
      <array><name>ReleaseDate</name><string>1989</string></array>
      <var><name>NumTracks</name><data>12</data></var>
    </struct>
```

Continued ↓

```
    <struct>
      <index>3</index>
      <var><name>TitleID</name><data>12328612</data></var>
      <array><name>Artist</name><string>Buffet, Jimmy</string></array>
      <array><name>Title</name><string>A-1-A</string></array>
      <array><name>Copyright</string><data>MCA</string></array>
      <array><name>Label</name><string>MCA</string></array>
      <array><name>ReleaseDate</name><string>1974</string></array>
      <var><name>NumTracks</name><data>11</data></var>
    </struct>
  </array>
</rsHeader>
```

- The **<rsHeader></rsHeader>** pair indicates which query this represents: it is simply an echo of the hdr CGI parameter.
- The **<struct></struct>** pair contains <var></var> pairs which contain the **Start**,**PageSize**, **NumberRecords**, **TotalRecords** and **PanelIndex** variables. The value of these variables, wrapped in the <data></data> pairs, provide information about the quantity of records.
- The **<array></array>** pair indicates that an array is to follow. Each record is wrapped in a <struct></struct> pair and the columns of each record are represented by the <var></var> pairs.
- The **<index></index>** pair represents the index into the array that this structure or record represents. String data is enclosed by <array></array> tags, where the data is enclosed in <string></string> tags. Numerical data is enclosed in <var></var> tags, where the data is enclosed in <data></data> tags. If the abs CGI parameter has been set, these would be absolute indexes (in this case, 51,52 and 53 instead of 1, 2 and 3).

A script error may be present. In this case, the error will be reported in the following XML format:

```
<rsHeader>
  <scriptError>
    <errorNumber>ErrorNumber</errorNumber>
    <errorDescription>ErrorDescription</errorDescription>
  </scriptError>
</rsHeader>
```

- **ErrorNumber** is a singled 32-bit value representing an error.
- **ErrorDescription** is a string describing the error.

The errors returned by the script may be generated by the underlying ADO technology used to access the database. In all cases, the **ErrorDescription** contains information about how to fix the error.

Creating an ASP file

All you need to do in your ASP file is include the NetLinxDBInclude.asp file, and call a single function, **`RunDBQuery`**. There are only two lines of code:

```
<!-- #INCLUDE FILE="NetlinxDBInclude.asp" -->
<% RunDBQuery "DeluxeCD.mdb", "" %>
```

The first parameter is the name of a Microsoft Access database. The second parameter is used for advanced applications and normally should be an empty string. The `RunDBQuery` function will take care of making the connection to the database and returning all the records to the NetLinx system. If you want to use a DSN entry instead of a Microsoft Access database, simply include the DSN= and the DSN entry name. The following syntax connects to a database setup using the DSN entry MyDSNEntry:

```
<!-- #INCLUDE FILE="NetlinxDBInclude.asp" -->
<% RunDBQuery "DSN=MyDsnEntry", "" %>
```

For more information about setting up a data source name (DSN), see the *Data Source Name* section on page 5.

Creating an AXI file

Now that the ASP file is complete, it's time to program the NetLinx AXI file. You need to include **NetlinxDBInclude.axi** in order to provide access to the support functions that make your job much easier. In the process of writing the AXI file, you will need to generate SQL statements to access the database. If you are not familiar with SQL, you have a couple of options. You can learn SQL, use Microsoft Access to generate SQL statements for you, or use DBWizard to help generate the SQL statements and the NetLinx code. The DBWizard option is the easiest and fastest way to integrate to the database. To learn more about SQL, return to the *Structure Query Language (SQL)* section on page 5.

The first part of the AXI file includes the NetlinxDBInclude.axi file and defines an IP device number for the connection to the web server:

```
#INCLUDE 'NetlinxDBInclude.axi'
(***********************************************************)
(*          DEVICE NUMBER DEFINITIONS GO BELOW             *)
(***********************************************************)
DEFINE_DEVICE

dvDB_CLIENT = 0:3:0
```

The next section creates NetLinx Data structures that will be used to hold the data returned from the database. These structures closely match the column names and data types that are defined in the database. This structure represents the titles table and contains information about each CD in the database:

Continued ↓

```
(***********************************************************)
(*                TYPE DEFINITIONS GO BELOW                *)
(***********************************************************)
DEFINE_TYPE

(* TITLES *)
STRUCTURE _sDB_TITLES
{
  CHAR        strArtist[128]
  CHAR        strCopyright[128]
  CHAR        strLabel[128]
  CHAR        strReleaseDate[128]
  CHAR        strTitle[128]
  SLONG       slTitleID
}
```

The next section creates NetLinx variables needed to store the data and connect to the database. The **sDB_CLIENT** variable contains information about the current database transaction, the location of the web server database script. The **sDB_QUE** variable keeps track of all database request and makes sure the script is sent one request at a time. The **sDB_TITLES** variable is an array of the **sDB_TITLES** structure to hold the data from the database. In order to keep track of the discs in the window, an **sDB_LISTBOX** structure is needed.

```
(***********************************************************)
(*               VARIABLE DEFINITIONS GO BELOW             *)
(***********************************************************)
DEFINE_VARIABLE


(* QUE VARS *)
VOLATILE _sDB_CLIENT  sDB_CLIENT     (* DB CLIENT *)
VOLATILE _sDB_QUE     sDB_QUE        (* dB QUE *)


(* DATABASE STRUCTURES *)
_sDB_TITLES sDB_TITLES[10]           (* TITLES *)
_sDB_LISTBOX sTitleListBox
```

The next section creates four calls to delete, add, update and read information in the database. It is here that we need our SQL and database setup information. When building SQL statements, you need to know the names of the tables and/or view and usually the names of any columns you are interested in reading or writing.

Another key piece of information is which column(s) in a table are the primary keys. Each record in a database usually has a unique column called the primary key. This field is unique for this entry in a given table. When deleting or updating a record in a database, you should reference the primary key of the table to make certain it is the record you are interested in modifying. **TitleID** is the primary key in our example.

When adding to a database, we may also need to provide a value for the primary key. In this case, you must supply a key that is not currently used. Many databases offer a column typed called **AutoNumber**. This is usually used as a unique key that automatically increments every time you add a record to the database. When using `AutoNumber` types for your primary key, you do not need to specify the value since it is automatically generated.

All of these calls generate an SQL statement to run against the database. Some people may find this part the hardest since SQL may be a new language to many people. If you are having trouble with SQL, refer to the *Structure Query Language (SQL)* section on page 5.

Once the SQL has been generated, the request is entered into a queue to be sent to the web server. The parameters for **DB_ADD_TO_QUE** are the queue structure for the connection, a tag, the SQL statement and a list box structure. The tag will be returned by the web server to identify the results. Tags should be unique for every query or sets of queries generated. (In this case, the tags are shared for all functions that operate on the same table.) The list box structure tells the server how many records we want to read starting from what position. Normally, you only need to read one page at a time and the list box routines manage that. The list box value has no effect if you are not reading from the database.

You may notice the calls to **DB_STRING_REPLACE**. These calls are used to replace any single quotes with two single quotes. SQL represents a single quote as two single quotes, like Axcess and NetLinx do.

Sometimes, the column names contains other characters that SQL considers invalid, like spaces and dashes. If you are using column names with these characters or continually get SQL syntax errors, try wrapping the column names in a pair of brackets **[ ]**. This is SQL's way of handling invalid characters.

```
(*****************************************)
(* NAME: DB_DELETE_TITLES                *)
(*****************************************************************************)
(* Format query to delete an entry of TITLES data                            *)
(*****************************************************************************)
DEFINE_FUNCTION DB_DELETE_TITLES(SLONG       slTitleID)
STACK_VAR
CHAR strSQL[1000]
_sDB_LISTBOX sTempListBox
{
  (* BUILD A QUERY STRING *)
  strSQL = "'DELETE FROM Titles WHERE TitleID = ',ITOA(slTitleID)"

  (* SEND THE QUERY *)
  DB_ADD_TO_QUE(sDB_QUE,'TITLES',strSQL,sTempListBox)
}
(*****************************************)
(* NAME: DB_ADD_TITLES                   *)
(*****************************************************************************)
(* Format query to add an entry to TITLES data                               *)
(*****************************************************************************)
DEFINE_FUNCTION DB_ADD_TITLES(CHAR        strArtist[128],
                              CHAR        strCopyright[128],
                              CHAR        strLabel[128],
                              CHAR        strReleaseDate[128],
                              CHAR        strTitle[128],
                              SLONG       slTitleID)
STACK_VAR
 CHAR strSQL[1000]
_sDB_LISTBOX sTempListBox
{
  (* ESCAPE ANY SINGLE QUOTES *)
  strArtist = DB_STRING_REPLACE(strArtist,"39","39,39") (* ' FOR '' *)
  strCopyright = DB_STRING_REPLACE(strCopyright,"39","39,39") (* ' FOR '' *)
  strLabel = DB_STRING_REPLACE(strLabel,"39","39,39") (* ' FOR '' *)
  strReleaseDate = DB_STRING_REPLACE(strReleaseDate,"39","39,39") (* ' FOR ''
*)
  strTitle = DB_STRING_REPLACE(strTitle,"39","39,39") (* ' FOR '' *)

  (* BUILD A QUERY STRING *)
  strSQL = "'INSERT INTO Titles (Artist, Copyright, Label, ReleaseDate, Title,
TitleID) ',
```

Continued ↓

```
            ' VALUES (',$27,strArtist,$27,', ',$27,strCopyright,$27,',
',$27,strLabel,$27,', ',
            $27,strReleaseDate,$27,', ',$27,strTitle,$27,',
',ITOA(slTitleID),')'"

  (* SEND THE QUERY *)
  DB_ADD_TO_QUE(sDB_QUE,'TITLES',strSQL,sTempListBox)
}
(****************************************)
(* NAME: DB_UPDATE_TITLES               *)
(*************************************************************************)
(* Format query to update an entry to TITLES data                        *)
(*************************************************************************)
DEFINE_FUNCTION DB_UPDATE_TITLES(CHAR        strArtist[128],
                                 CHAR        strCopyright[128],
                                 CHAR        strLabel[128],
                                 CHAR        strReleaseDate[128],
                                 CHAR        strTitle[128],
                                 SLONG       slTitleID)
STACK_VAR
CHAR strSQL[1000]
_sDB_LISTBOX sTempListBox
{
  (* ESCAPE ANY SINGLE QUOTES *)
  strArtist = DB_STRING_REPLACE(strArtist,"39","39,39") (* ' FOR '' *)
  strCopyright = DB_STRING_REPLACE(strCopyright,"39","39,39") (* ' FOR '' *)
  strLabel = DB_STRING_REPLACE(strLabel,"39","39,39") (* ' FOR '' *)
  strReleaseDate = DB_STRING_REPLACE(strReleaseDate,"39","39,39") (* ' FOR ''
*)
  strTitle = DB_STRING_REPLACE(strTitle,"39","39,39") (* ' FOR '' *)
  (* BUILD A QUERY STRING *)
  strSQL = "'UPDATE Titles SET Artist = ',$27,strArtist,$27,', Copyright = ',
           $27,strCopyright,$27,', Label = ',$27,strLabel,$27,', ReleaseDate =
',
           $27,strReleaseDate,$27,', Title = ',$27,strTitle,$27,', TitleID = ',
            ITOA(slTitleID),' WHERE TitleID = ',ITOA(slTitleID)"

  (* SEND THE QUERY *)
  DB_ADD_TO_QUE(sDB_QUE,'TITLES',strSQL,sTempListBox)
}
(****************************************)
(* NAME: DB_READ_TITLES                 *)
(*************************************************************************)
(* Format query to read a page of TITLES data                            *)
(*************************************************************************)
DEFINE_FUNCTION DB_READ_TITLES(_sDB_LISTBOX sTempListBox,CHAR strSEARCH[])
STACK_VAR
CHAR    strSQL[1000]
{
  (* BUILD A QUERY STRING *)
  strSQL = "'SELECT * FROM Titles'"
  IF (LENGTH_STRING(strSEARCH))
    strSQL = "strSQL,' WHERE ',strSEARCH"
  strSQL = "strSQL,' ORDER BY Artist'"

  (* SEND THE QUERY *)
  DB_ADD_TO_QUE(sDB_QUE,'TITLES',strSQL,sTempListBox)
}
```

The next section initializes the connection to the database script and creates a buffer to hold the results. The call to **DB_INIT_CLIENT** initializes the **sDB_CLIENT** structure created in the variable section. The parameters are the: IP address of the web server, IP port of the web server (usually **80**) and path to the database script. This path is the directory and file name of the ASP file. You will

need to fill this out properly. See the section *Putting It All Together* section on page 28 for reference. Also, call DB_LISTBOX_INIT to initialize your listbox.

```
(*************************************************************)
(*                  STARTUP CODE GOES BELOW                  *)
(*************************************************************)
DEFINE_START


(* INIT dB WEB CLIENT *)
DB_INIT_CLIENT (sDB_CLIENT,dvDB_CLIENT,'192.168.12.175',80,'/dB/CDExample.asp')
CREATE_BUFFER dvDB_CLIENT,sDB_CLIENT.strBUFF
(*************************************************************)
(*                  STARTUP CODE GOES BELOW                  *)
(*************************************************************)
DEFINE_START

DB_LISTBOX_INIT(sTitleListBox,10,1)
```

The next section of code reads the information from the web server and copies the data to the structure you created to hold the results. The **DATA_EVENT** below has a couple of different parts.

The **ONERROR** section of the DATA_EVENT occurs whenever we encounter a communications problem. The code here simply clears the buffer, calls **DB_GET_IP_ERROR** to return a text error from our error code (held by **DATA.NUMBER**) and prints the error to the terminal.

The **ONLINE** section of the DATA_EVENT occurs whenever a connection to the web server is established. The code here clears our buffer and calls **DB_BUILD_HTTP_GET** to format the request for the web server. Send this request to the server and it will process the results and return the information.

The **OFFLINE** section of the DATA_EVENT occurs whenever a connection to the web server is dropped. A web server will normally drop the connection whenever is has finished processing the request, so this is a great time for to interpret the results received. Normally, this would occur in a **STRING** section of a DATA_EVENT. The first thing to do is create a series of variables that needed through the event. To kick off the processing, call **DB_PROCESS_HTTP_HEADER** which will parse the HTTP header information from the response and deal with any cookies the web server wants you to deal with. Once that has happened, drop into a select active to process the data.

The first active in your select active makes sure the HTTP return code is **200**. If not, some error has occurred but luckily your call to **DB_PROCESS_HTTP_HEADER** has already processed all the errors and printed those errors to terminal. This call will detect standard HTTP errors as well as ASP script errors. ASP script errors may be generated by incorrect SQL so if something does not appear to work, check the terminal for a description of the problem.

If DB_PROCESS_HTTP_HEADER has returned a value of 200 (this is a HTTP code meaning OK), then there is data for to process. The first thing to do is look for your tag. This is the tag you sent with the request with a few extra characters added. The first is the **</rs** which signifies then end of an XML tag whose name is **rs<Our Tag>** where rs stands for record set. The trailing > is the end of the XML tag. One of the things our call to DB_PROCESS_HTTP_HEADER did for you was find the start of the XML packet. This is the point from which you begin your search.

Once we have identified what the response is to, we can process it. The next thing we do is pull out the list box information contained in the XML by calling **DB_PROCESS_LISTBOX**. This function returns the position to begin to parsing for more information. You should expect to see a series of structures that represent your data. Simply loop as long as you can find **<struct>** in the XML

packet, isolate the **<struct></struct>** package and remove the data contained within. The helper function **DB_GET_XML_VALUE** removes the data for a given field name and returns the string containing the data. All you need to know is the column name from that database and this function returns the data. If the data is not of string type, you need to convert that data to the acceptable type using a call to **ATOI**, **ATOL** or **ATOF**. Once we have processed all the data from the XML, call a **DISPLAY_** function to display the data.

```
(***********************************************************)
(*            EVENTS GO BELOW                              *)
(***********************************************************)
DEFINE_EVENT

(*********************)
(*** dB CLIENT URL ***)
(*********************)
DATA_EVENT [dvDB_CLIENT]
{
  ONERROR:                          (* EVENT - ERROR *********************)
  {
    (* CLEAR BUFFER *)
    CLEAR_BUFFER sDB_CLIENT.strBUFF
    SEND_STRING 0,dB_GET_IP_ERROR (DATA.NUMBER)
  }
  ONLINE:                           (* EVENT - ONLINE ********************)
  {
    (* SEND THE HTTP GET TO THE WEB SERVER... *)
    CLEAR_BUFFER sDB_CLIENT.strBUFF
    SEND_STRING dvDB_CLIENT, dB_BUILD_HTTP_GET(sDB_CLIENT)
  }
  OFFLINE:                          (* EVENT - OFFLINE *******************)
  {
    STACK_VAR
    LONG         lCODE                (* HTTP RETURN CODE *)
    INTEGER      nLIST_PTR            (* INDEX FOR RECORDS *)
    LONG         lLOOP                (* LOOPS OVER RECORDS *)
    INTEGER      nFIRST               (* POSITION OF START OF RECORD *)
    INTEGER      nLAST                (* POSITION OF END OF RECORD *)
    INTEGER      nCOUNT               (* CHARACTERS IN RECORD *)
    CHAR         strDB_RECORD[2000]   (* HOLDS A RECORD *)
    CHAR         strDB_ITEM[100]      (* ITEM OF RECORD *)
    _sDB_LISTBOX sTempListBox         (* A LIST BOX *)

    (* GET COOKIE? *)
    lCODE = DB_PROCESS_HTTP_HEADER(sDB_CLIENT)
    SELECT
    {
      (* Bad Webserver return *)
      ACTIVE (lCODE <> 200): {}
      (*  TITLES *)
      ACTIVE(FIND_STRING(sDB_CLIENT.strBUFF,'</rsTITLES>',sDB_CLIENT.lHTML_START)):
      {
        (* Found an , so parse it for data *)
        lLOOP = DB_PROCESS_LISTBOX (sDB_CLIENT,sTempListBox)
        nLIST_PTR = 0
        WHILE(lLOOP)
        {
          nFIRST = FIND_STRING(sDB_CLIENT.strBUFF,'<struct>',lLOOP)
          nLAST  = FIND_STRING(sDB_CLIENT.strBUFF,'</struct>',lLOOP)
          nCOUNT = nLAST - nFIRST + LENGTH_STRING('</struct')
          IF (nFIRST)
          {
            nLIST_PTR = nLIST_PTR + 1
            strDB_RECORD = MID_STRING(sDB_CLIENT.strBUFF, nFIRST, nCOUNT)
```

Continued

```
            sDB_TITLES[nLIST_PTR].strArtist =
DB_GET_XML_VALUE(strDB_RECORD,'Artist')
            sDB_TITLES[nLIST_PTR].strCopyright =
DB_GET_XML_VALUE(strDB_RECORD,'Copyright')
            sDB_TITLES[nLIST_PTR].strLabel =
DB_GET_XML_VALUE(strDB_RECORD,'Label')
            sDB_TITLES[nLIST_PTR].strReleaseDate =
DB_GET_XML_VALUE(strDB_RECORD,'ReleaseDate')
            sDB_TITLES[nLIST_PTR].strTitle =
DB_GET_XML_VALUE(strDB_RECORD,'Title')
            sDB_TITLES[nLIST_PTR].slTitleID =
ATOL(DB_GET_XML_VALUE(strDB_RECORD,'TitleID'))
          }
          lLOOP =
FIND_STRING(sDB_CLIENT.strBUFF,'<struct>',nLAST+LENGTH_STRING('<struct'))
        }
        IF (sTempListBox.snTOTAL > 0) sTempListBox.snTOTAL =
TYPE_CAST(nLIST_PTR)
        DB_DISPLAY_TITLES(sDB_TITLES,sTempListBox)
      }
    }

    (* CLEAR BUFFER *)
    DB_ACK_QUE (sDB_QUE,sDB_CLIENT)
    CLEAR_BUFFER sDB_CLIENT.strBUFF
  }
}
```

The very last bit of code keeps your queue going. Call `DB_CHECK_QUE` in mainline to make sure all request get processed and sent to the web server.

```
(***********************************************************)
(*             THE ACTUAL PROGRAM GOES BELOW               *)
(***********************************************************)
DEFINE_PROGRAM

DB_CHECK_QUE(sDB_QUE,sDB_CLIENT)
```

Creating an AXS file

The CDExample.axs file contains the remaining part of the code. You still need to provide the user with some way to read and possibly update all this information. The first thing to do is create a touch panel device to use as a user interface and include your AXI file.

```
(***********************************************************)
(*          DEVICE NUMBER DEFINITIONS GO BELOW             *)
(***********************************************************)
DEFINE_DEVICE

dvTP          = 128:1:0


(***********************************************************)
(*               INCLUDE FILES GO BELOW                    *)
(***********************************************************)
#INCLUDE 'CDList.axi'
```

The next section creates some variables you will need. Create a DEVCHAN set to hold the buttons which control the movement through the list. This action can be seen when you get to the DEFINE_EVENT section below.

```
(***********************************************************)
(*               VARIABLE DEFINITIONS GO BELOW             *)
(***********************************************************)
DEFINE_VARIABLE


_sDB_LISTBOX sTitleListBox


DEVCHAN dcTITLES_CTRL[] = {
                            { dvTP,11 }, { dvTP,12 }, { dvTP,13 }, { dvTP,14 },
{ dvTP,15 }
                          }
```

The next section creates a display call to display our results. This function is called whenever **TITLE** data is processed from the server in your **DATA_EVENT**. An array of structures will be passed containing the data and an **_sDB_LISTBOX** structure. The listbox structure will contain information about the number of records and from what location they have started.

Basically, loop over the valid number of records and display the data you are interested in. There are two Send_Command's that send Artists and CD Title information to our touch panel. The next loop is used for clean up. Continue looping up to the display size, which is the total number of items that can be displayed, and clear out the variable text. The last bit of code uses information contained in your listbox. The first Send_Command displays record information in the form displaying 1 - 10 of 100. The listbox also has a value you can send to a slider to represent the current position in the list, like a scroll bar. The last code copies the listbox to the listbox variable you defined. A template for this function is generated by DBWizard and included in the include file. The easiest way to write this function is to copy it from the .axi file and add any customizations required.

```
(***********************************************************)
(*           SUBROUTINE DEFINITIONS GO BELOW               *)
(***********************************************************)

(*****************************************)
(* NAME: DB DISPLAY TITLES                   *)
(*****************************************************************************)
(* Display a page of TITLES data                                             *)
(*****************************************************************************)
DEFINE_FUNCTION DB_DISPLAY_TITLES(_sDB_TITLES sDB_TITLES[],_sDB_LISTBOX
sTempListBox)
LOCAL_VAR
INTEGER   nLOOP
{
  (* The following are elements of the structure where nLOOP=1 to nCOUNT: *)
  FOR (nLOOP = 1; nLOOP <= sTempListBox.nCOUNT; nLOOP++)
  {
    (*
    SEND_STRING 0,"'strArtist=',sDB_TITLES[nLOOP].strArtist"
    SEND_STRING 0,"'strCopyright=',sDB_TITLES[nLOOP].strCopyright"
    SEND_STRING 0,"'strLabel=',sDB_TITLES[nLOOP].strLabel"
    SEND_STRING 0,"'strReleaseDate=',sDB_TITLES[nLOOP].strReleaseDate"
    SEND_STRING 0,"'strTitle=',sDB_TITLES[nLOOP].strTitle"
    SEND_STRING 0,"'slTitleID=',ITOA(sDB_TITLES[nLOOP].slTitleID)"*)
    (* ADD DISPLAY CODE HERE *)
    SEND_COMMAND dvTP,"'!T',0+nLOOP,sDB_TITLES[nLOOP].strArtist"
    SEND_COMMAND dvTP,"'!T',10+nLOOP,sDB_TITLES[nLOOP].strTitle"
```

Continued

```
  }
  FOR (; nLOOP <= sTempListBox.nDISPLAY_SIZE; nLOOP++)
  {
    (* CLEAN UP EMPTY ENTRIES HERE *)
    SEND_COMMAND dvTP,"'!T',0+nLOOP,''"
    SEND_COMMAND dvTP,"'!T',10+nLOOP,''"
  }
  SEND_COMMAND dvTP,"'!T',49,'Displaying ',ITOA(sTempListBox.snFIRST),'-',
                            ITOA(sTempListBox.snLAST),' of
',ITOA(sTempListBox.snTOTAL)"
  SEND_LEVEL dvTP,1,sTempListBox.nLEVEL_VAL
  sTitleListBox = sTempListBox
  }
```

The next section creates a level for the scroll bar. Use this later to actively jump to any point in your list.

```
(***********************************************************)
(*                STARTUP CODE GOES BELOW                  *)
(***********************************************************)
DEFINE_START


(* Get Level for active slider *)
CREATE_LEVEL dvTP,1,nTITLE_LVL
```

The next section handles the request from the touch panel to manipulate your list. The first two buttons are up and down. Use **DB_LISTBOX_UP** and **DB_LISTBOX_DOWN** to manipulate your list box. The next two buttons jump to the beginning and end of the list. Call **DB_LISTBOX_SET** with a value of **1** to go to the beginning and a value of **$FFFF** to go to the end. The value `$FFFF` is used here as a very large value. Anytime you try to set past the end of the list, `DB_LISTBOX_SET` will automatically make sure the selected value is inside the selected list. Using `$FFFF` is an easy way to force the function to automatically position to the end of the list. The last button is from your scroll bar slider. Notice you capture the position on the release after you have stopped moving the level and lifted your finger off the slider. Send the level you created earlier to **DB_LISTBOX_SET** and set the last parameter, `bFROM_SLIDER`, to 1. **DB_LISTBOX_SET** will automatically scale the value and select the position in the list that matches your request.

After you have adjusted your listbox, call **DB_READ_TITLES** to build your SQL and send the request to the server. When the response from this request comes in, your **DB_DISPLAY_TITLES** function will be called and our requests will be displayed. A template for calling this function was generated by DBWizard and included in the include file. You can copy the call template from the include file and paste it into your code.

```
(***************************************)
(* DEVCHAN: Title Control              *)
(***************************************)
BUTTON_EVENT[dcTITLES_CTRL]
{
  PUSH:
  {
    STACK_VAR nIDX
    nIDX = GET_LAST(dcTITLES_CTRL)
    SWITCH (nIDX)
    {
      CASE 1: DB_LISTBOX_UP (sTitleListBox)
      CASE 2: DB_LISTBOX_DOWN (sTitleListBox)
      CASE 3: DB_LISTBOX_SET (sTitleListBox,1,0)
      CASE 4: DB_LISTBOX_SET (sTitleListBox,$FFFF,0)
    }
  DB_READ_TITLES(sTitleListBox,"")
```

Continued ↓

```
    TO[dcTITLES_CTRL[nIDX]]
  }
  }
```

If you want to edit the database, you need to capture data from the user and call your other SQL building routines to send your request off to the server.

NetlinxDBInclude.asp

Functions

The following table describe the functions contained in the NetlinxDBInclude.asp file.

NetlinxDBInclude.asp Functions	
XMLTag Converts field name to XML compatible tag.	The XMLTag function is used when converting record sets to XML. Syntax: `XMLTag(szStr)` Variable: szStr = Represents the string to be converted to XML tag. (Required.) Return Values: XMLTag returns the string converted to XML tag.
CDATAit Converts XML Parameter to CDATA format.	The CDATAit function is used when converting record sets to XML. Syntax: `CDATAit(strData)` Variables: strData = Represents the data to be wrapped in CDATA. (Required.) Return Values: CDATAit returns srtData wrapped in CDATA.
GetXMLFromADORS Converts ADO Recordset to XML.	The GetXMLFromADORS function is used to convert record sets to XML. Syntax: `GetXMLFromADORS(objADORS , strHdr, nStart, nPageSize)` Variable: objADORS = Represents the record set object to be converted. (Required.) strHdr = Represents the header to attached to XML record set. (Required.) nStart = Represents the starting record number. (Required.) nPageSize = Represents the number of records to retrieve. (Required.) Return Values: The GetXMLFromADORS function is used to convert record sets to XML.

Continued

NetlinxDBInclude.asp Functions (Cont.)

RunDBQuery Converts ADO RS to XML.	The RunDBQuery function is used to convert record sets to XML. Syntax: `RunDBQuery( strDBPath, strProvider )` Variables: strDBPath = Represents the file path or DSN to the database. If the file has no path included, the path is assumed to be local in the same directory as the ASP file. (Required.) strProvider = Represents the provider string for the database. No value is needed for a DNS type connection. If the file contains ".mdb", a provider of "Microsoft.Jet.OLEDB.4.0" is added to the one not provided. (Required.) Return Values: RunDBQuery returns an XML representation of a record set.
ErrorToXML Converts error code and descriptions to XML.	The ErrorToXML function is used to convert errors to XML. Syntax: `ErrorToXML(nNumber, strDesc)` Variables: nNumber = Represents the error number. (Required.) strDesc = Required. Represents the error description. (Required.) Return Values: XML representation of an error.
VariableHeader Creates a variable header (hungarian notation) for a given ADO variable type.	The VariableHeader is used to add the hungarian notation to field names so they match the NetLinx variable definition designed to hold the data from a given field. Syntax: `VariableHeader(ADOVarType)` Variables: nADOVarType = Represents the ADO variable type. (Required.) Return Values: A string to add to the field name to create a NetLinx variable name.

NetlinxDBInclude.axi

Constants

All constants can be overridden by defining your own values in your program.

NetlinxDBInclude.axi Constants

Parameter	Value	Description	Notes
IP_TCP	= 1	(* TCP/IP COMMUNICATIONS *)	Passed IP_CLIENT_OPEN and IP_CLIENT_CLOSED.
nDB_MAX_TIMEOUT	= 60	(* MAXIMUM TO WAIT FOR XML RESPONSE *)	Time to wait for ASP to process and return XML.
nDB_MAX_BUFF_SIZE	= 10000	(* WEB SERVER BUFFER SIZE *)	Buffer size for XML.
nDB_MAX_URL_SIZE	= 1000	(* WEB SERVER URL SIZE *)	URL size and therefore, SQL statements.
nDB_MAX_HDR_SIZE	= nDB_MAX_URL_SIZE +200	(* WEB SERVER HTTP GET HEADER SIZE *)	GET header size. 200 bytes above the max URL size.
nDB_MAX_PARAM_SIZE	= 300	(* XML PARAMETER SIZE *)	Maximum size of XML values and ASP cookie.
nDB_MAX_QUE_SIZE	= 10	(* WEB SERVER QUE SIZE *)	Number of commands to allow in queue.
nDB_LISTBOX_PAGE_OVERLAP	= 0	(* LIST BOX PAGE OVERLAP *)	Control whether a new list box page contains 1 of entry for reference

Variables

The following table is a list of variables for NetlinxDBInclude.axi.

NetlinxDBInclude.axi Variables

Parameter	Value	Description	Notes
IP_ADDRESS_STRUCT	sMY_IPAddress	(* MY IP ADDRESS *)	Holds the IP address of the Master.
CHAR	bDB_DEBUG	(* SET TO 1 TO DEBUG *)	Set to 1 to get debug info from the NetlinxDBInclude.axi.

The include file will need to be included before you variable section in order to define variables to _sDB_LISTBOX type. However, if you wait until the variable section to define debug, it will already be defined. The best way to solve this problem is to include NetlinxDBInclude.axi just before the DEFINE_VARIABLE section and define bDB_DEBUG as a constant and set it to 1.

Structures

```
STRUCTURE _sDB_CLIENT
{
    CHAR        strBUFF[10000]          (* BUFFER FOR XML *)
    CHAR        strQUERYSTRING[1000]    (* QUERY STRING *)
    CHAR        strWEB_SERVER[100]      (* IP OR NAME OF SERVER *)
    CHAR        strDB_ASP_FILE[100]     (* FILE NAME/PATH OF DB ASP FILE *)
    INTEGER     nWEB_PORT               (* PORT WEB SERVER IS LISTENING ON *)
    CHAR        strASP_COOKIE[300]      (* ASP COOKIE *)
    DEV         dvSOCKET                (* LOCAL PORT HANDLE/DEVICE *)
    INTEGER     nMAX_TIMEOUT            (* MAXIMUM TIMEOUT IN SECONDS *)
    INTEGER     nTO_COUNT               (* TIMEOUT COUNT *)
    CHAR        bTO_LO                  (* TIMEOUT LOCKOUT *)
    LONG        lHTML_START             (* POSITION OF HTML SEQUENCE *)
    CHAR        nVERSION_CHECK          (* 1 IF WE CHECKED VERSION *)
}

STRUCTURE _sDB_QUE
{
    CHAR        strQUEUE[10][2000]      (* QUEUED COMMANDS *)
    INTEGER     nQ_HEAD                 (* QUEUE HEAD POINTER *)
    INTEGER     NQ_TAIL                 (* QUEUE TAIL POINTER *)
    CHAR        bQ_HAS_ITEMS            (* 1 IF ANY ITEMS ARE IN THE QUEUE *)
    CHAR        bQ_READY                (* 1 IF READY TO SEND THE NEXT CMD *)
    INTEGER     NQ_MAX                  (* MAXIMUM ENTRIES IN QUE *)
    CHAR        bINIT                   (* 1 IF INIT *)
}

STRUCTURE _sDB_HTTP_HEADERS
{
    CHAR        strHTTP_PROT[20]        (* HTTP PROTOCOL VERSION *)
    LONG        lHTTP_CODE              (* HTTP RETURN CODE *)
    CHAR        strHTTP_DESC[100]       (* RETURN DESCRIPTION *)
    CHAR        strHTTP_SVR [100]       (* SERVER DESCRIPTION *)
    CHAR        strHTTP_DATE[100]       (* SERVER DATE *)
    CHAR        strHTTP_CTYPE[100]      (* CONTENT TYPE *)
    LONG        lHTTP_CLENGTH           (* CONTENT LENGTH *)
    CHAR        strHTTP_COOKIE[300]     (* COOKIE FROM SERVER *)
}

STRUCTURE _sDB_LISTBOX
{
    SINTEGER    snFIRST                 (* FIRST ENTRY IN THE LIST BOX DISPLAY *)
    SINTEGER    snLAST                  (* LAST ENTRY IN THE LIST BOX DISPLAY *)
    INTEGER     nDISPLAY_SIZE           (* NUMBER OF ITEMS TO LIST PER PAGE *)
    SINTEGER    snTOTAL                 (* TOTAL NUMBER OF ITEMS *)
    INTEGER     nLEVEL_VAL              (* LEVEL VALUES FOR SLIDER POSITION (0-255) *)
    INTEGER     nCOUNT                  (* COUNT OF ITEMS ON CURRENT PAGE *)
    INTEGER     nPNL_IDX                (* VALUE: PANEL INDEX *)
}
```

Functions

The following table is a list of functions contained in the NetlinxDBInclude.axi.

NetlinxDBInclude.asp Functions	
DB_ACK_QUE() Acknowledges the queue so the next message can be sent in the queue structure.	The DB_ACK_QUE function is used to acknowledge the last message sent by the queue. It should be called whenever a message is properly processed in the DATA_EVENT for the database server script connection. This function also closes the IP socket used by the sDB_CLIENT structure. Syntax: DB_ACK_QUE (_sDB_QUE sDB_QUE, _sDB_CLIENT sDB_CLIENT) Variable: sDB_QUE = Represents the sDB_QUE for a given database server script connection. (Required.) sDB_CLIENT = Represents the sDB_CLIENT for a given database server script connection. (Required.) Return Values: DB_ACK_QUE does not return a value.
DB_ADD_TO_QUE() Adds a message to the queue structure sDB_QUE.	The DB_ADD_TO_QUE function is used to send all database requests. Syntax: DB_ADD_TO_QUE(_sDB_QUE sDB_QUE, CHAR strHDR[], CHAR strSQL[], _sDB_Listbox sTempListBox) Variables: sDB_QUE = Represents the sDB_QUE for a given database server script connection. (Required.) srHDR = Required. Represents a header to be added to the XML response to identify the response. (Required.) strSQL = Required. Represents the SQL statement to be processed by the database server script. (Required.) sTempListBox = The listbox structure containing the start and page size information. (Required.) Return Values: • 0 if the message could not be added. • Otherwise, it returns the queue position the message was added to.
DB_BUILD_HTTP_GET() Converts the values from an sDB_CLIENT structure into a HTTP_GET request.	The DB_BUILD_HTTP_GET function is used to build the HTTP_GET request for a given database operation. Syntax: DB_BUILD_HTTP_GET (_sDB_CLIENT sDB_CLIENT) Variable: sDB_CLIENT = Represents the sDB_CLIENT for a given database server script connection. (Required.) Return Values: DB_BUILD_HTTP_GET returns the HTTP_GET request as a string to be sent to the web server.
DB_CHECK_MASTER_VERSION() Checks the Master's version and prints an error if XML_TO_VARIABLE is not supported.	Syntax: DB_CHECK_MASTER_VERSION() The DB_CHECK_MASTER_VERSION function syntax has no arguments. Return Values: • 1 if the Master supports the required version for XML_TO_VARIABLE. • 0 if it does not.

Continued

NetlinxDBInclude.asp Functions (Cont.)	
DB_CHECK_QUE() Sends the next command from the sDB_QUE to the sDB_CLIENT database server connection.	The DB_CHECK_QUE function is used to watch the queue and send messages when the client is ready. It should be called once in DEFINE_PROGRAM. Syntax: DB_BUILD_HTTP_GET (_sDB_CLIENT sDB_CLIENT) Variables: sDB_QUE = Represents the sDB_QUE for a given database server script connection. (Required.) sDB_CLIENT = Represents the sDB_CLIENT for a given database server script connection. (Required.) Return Values: • 1 if a message was processed. • 0 if not.
DB_GET_HTTP_ERROR() Formats a NetLinx Socket error and returns the formatted string.	The DB_GET_HTTP_ERROR is used to print descriptive errors for HTTP return codes. This function is a bit misnamed since it prints HTTP return code descriptions, all of which are not errors. Syntax: DB_GET_HTTP_ERROR(LONG lCODE) Variable: lCODE = Represents the HTTP return code. (Required.) Return Values: DB_GET_HTTP_ERROR returns a string formatted for the error code in the format: HTTP_ERROR (<error code>): <error description>.
DB_GET_HTTP_HEADERS Reads the HTTP headers from a stream, and copies them to an sDB_HTTP_HEADERS structure.	Syntax: DB_GET_HTTP_HEADERS(CHAR strHTML[], LONG lCUR_POS, _sDB_HTTP_HEADERS sHEADERS) Variable: strHTML = HTML stream. (Required.) lCUR_POs = Position in HTML to begin parsing. (Required.) sHEADERS = Structure to copy headers to. (Required.) Return Values: The function does not return a value. However, lCUR_POS will contain the next available HTML character and sHEADERS will contain all header information contained in the HTML stream that overlaps with members of sDB_HTTP_HEADERS. Remarks: The DB_GET_HTTP_HEADERS is used to process HTTP headers. It is called by DB_PROCESS_HTTP_HEADER().
DB_GET_IP_ERROR Formats a NetLinx Socket error and returns the formatted string.	Syntax: DB_GET_IP_ERROR(LONG lERR) Variable: lERR = Represents the NetLinx error code from DATA_NUMBER. (Required.) Return Values: DB_GET_IP_ERROR returns a string formatted for the error code in the format: IP ERROR (<error code>): <error description> (<applicable function>).

Continued ↓

NetlinxDBInclude.asp Functions (Cont.)	
DB_GET_XML_VALUE() Extracts values from XML into NetLinx data structures only when not using XML_TO_VARIABLE.	The DB_GET_XML_VALUE function is used to extract values from XML into NetLinx data structures only when not using XML_TO_VARIABLE. Syntax: DB_GET_XML_VALUE(CHAR strITEM_DATA[], CHAR strTAG[]) Variable: strITEM_DATA = Represents the XML stream containing the data to be extracted. (Required.) strTAG = Represents the XML tag from which the data should be extracted. (Required.) Return Values: DB_GET_XML_VALUE returns the XML value for the supplied tag.
DB_INIT_CLIENT() Extracts a database script error from an sDB_CLIENT structure and prints it to the NetLinx terminal.	The DB_INIT_CLIENT function is used to initialize the sDB_CLIENT structure. It should be called once in DEFINE_START. Syntax: DB_INIT_CLIENT (_sDB_CLIENT sDB_CLIENT, DEV dvSKT, CHAR strWS[], INTEGER nPORT, CHAR strFILE[]) Variables: sDB_CLIENT = Represents the sDB_CLIENT for a given database server script connection. (Required.) dvSKT = Represents the IP device for a given database server script connection. (Required.) strWS = Represents the IP address or host name for the web server for a given database server script connection. (Required.) nPORT = Represents the IP port of the web server for a given database server script connection. If 0 is supplied, 80 will be used. (Required.) strFILE = Represents the Path and File of the database server script relative to the www root of the web server for a given database server script connection. If the path does not start with "/", a "/" is added. (Required.) Return Values: DB_INIT_CLIENT does not return a value.
DB_IS_TRUE() Converts the string "TRUE" and "FALSE" into 1 or 0, respectively.	The DB_IS_TRUE function is used to convert Boolean parameters from XML into CHAR types in NetLinx. Syntax: DB_IS_TRUE (CHAR strSTR[]) Variable: strSTR = Represents the string containing TRUE or FALSE. (Required.) Return Values: • 1 if the string contains TRUE. • 0 if the string does not contain TRUE. The search is case-insensitive.
DB_LISTBOX_DOWN() Sets the starting index of an sDB_LISTBOX structure to display the next page.	The DB_LISTBOX_DOWN moves the starting index to display the next page. The constant nDB_LISTBOX_PAGE_OVERLAP controls how the new index is calculated. If the constant is set to 1 (default), the last item in the list will become the first item in the list. If the constant is 0, the new page will not contain any items from the previous page. Syntax: DB_LISTBOX_DOWN(_sDB_LISTBOX sTempListBox) Variable: sTempListBox = Represents the list box structure. (Required.) Return Values: DB_LISTBOX_DOWN returns the starting position of the list box.

Continued

NetlinxDBInclude.asp Functions (Cont.)	
DB_LISTBOX_INIT() Initializes an sDB_LISTBOX structure.	The DB_LISTBOX_INIT should be called for all sDB_LISTBOX structures before use. Syntax: DB_LISTBOX_INIT(_sDB_LISTBOX sTempListBox, INTEGER DisplaySize, INTEGER nPanelIndex) Variables: sTempListBox = Represents the list box structure. (Required.) DisplaySize = Represents the size of page for the list box. (Required.) nPanelIndex = The index for the panel viewing data with this list box. (Required.) Return Values: DB_LISTBOX_INIT does not return a value.
DB_LISTBOX_SET() Sets the starting index of an sDB_LISTBOX structure.	The DB_LISTBOX_SET should return the same value as was passed in unless a limit was reached. Since the maximum size of a list box using this structure is 32767 (maximum size of a SINTEGER in NetLinx), sending any value larger than this will force the list box to the last page. Typically, $FFFF is the value used for this. If the bFROM_SLIDER flag is set, then the value is assumed to be 0 - 255 and will be scaled as a percentage of the total items in the list. Syntax: DB_LISTBOX_SET(_sDB_LISTBOX sTempListBox, INTEGER nIDX, CHAR bFROM_SLIDER) Variable: sTempListBox = Represents the list box structure. (Required.) nIDX = Represents the new starting index for the list box. (Required.) bFROM_SLIDER = Set to 1 if value is a raw (0 - 255) value from a slider. (Required.) Return Values: DB_LISTBOX_SET returns the starting position of the list box.
DB_LISTBOX_UP() Sets the starting index of an sDB_LISTBOX structure to display the previous page.	The DB_LISTBOX_UP moves the starting index to display the previous page. The constant nDB_LISTBOX_PAGE_OVERLAP controls how the new index is calculated. If the constant is set to 1 (default), the first item in the list will become the last item in the list. If the constant is 0, the new page will not contain any items from the previous page. Syntax: DB_LISTBOX_UP(_sDB_LISTBOX sTempListBox) Variable: sTempListBox = Represents the list box structure. (Required.) Return Values: DB_LISTBOX_UP returns the starting position of the list box.
DB_MAKE_URL_ STRING() Converts a string to a URL compatible string. All characters illegal in URL's are replaced with a "%" hex code equivalent.	The DB_MAKE_URL_STRING function is used to convert SQL queries into CGI strings. It uses DB_STRING_REPLACE. Syntax: DB_MAKE_URL_STRING(CHAR strSTR[]) Variable: strSTR = Represents the string to be converted by the process. (Required.) Return Values: DB_MAKE_URL_STRING returns the string strSTR after the all invalid characters have been replaced.

Continued

NetlinxDBInclude.asp Functions (Cont.)	
DB_PRINT_ERROR() Extracts a database script error from an sDB_CLIENT structure and prints it to the NetLinx terminal.	The DB_PRINT_ERROR function is used to print any database script errors to the NetLinx terminal. This should be called whenever the XML stream from a web server is processed. Syntax: DB_PRINT_ERROR (_sDB_CLIENT sDB_CLIENT) Variables: sDB_CLIENT = Represents the sDB_CLIENT for a given database server script connection. (Required.) Return Values: • 1 if an error was found. • 0 if no error was found.
DB_PRINT_HTML() Prints text from HTML to terminal.	The DB_PRINT_HTML is used to print descriptive HTTP errors. Syntax: DB_PRINT_HTML(CHAR strHTML[], LONG lCUR_POS) Variable: strHTML = Represents the HTML stream. (Required.) lCUR_POS = Represents the position in HTML to begin parsing. (Required.) Return Values: DB_PRINT_HTML does not return a value.
DB_PROCESS_HTTP_ HEADER() Extracts the HTTP Headers Cookie value from an sDB_CLIENT structure.	The DB_PROCESS_HTTP_HEADER function is used to parse the response from the server. DB_PROCESS_HTTP_HEADER looks for HTTP errors and prints a description of the problem. This should be called whenever the XML stream from a web server is processed. Syntax: DB_PROCESS_HTTP_HEADER (_sDB_CLIENT sDB_CLIENT) Variable: sDB_CLIENT = Represents the sDB_CLIENT for a given database server script connection. (Required.) Return Values: DB_PROCESS_HTTP_HEADER returns the HTTP return code. The cookie, if any, is also copied into the sDB_CLIENT structure.
DB_PROCESS_ LISTBOX() Retrieves the list box values from the XML stream and copies them to an sDB_LISTBOX structure.	The DB_PROCESS_LISTBOX copies the list box values from the XML to the sTempListBox structure. Syntax: DB_PROCESS_LISTBOX(_sDB_CLIENT sDB_CLIENT,_sDB_LISTBOX sTempListBox) Variable: sDB_CLIENT = Represents the sDB_CLIENT for a given database server script connection. (Required.) sTempListBox = Represents the list box structure. (Required.) Return Values: DB_PROCESS_LISTBOX returns the starting position of the <array> tag in the XML stream.

Continued

NetlinxDBInclude.asp Functions (Cont.)	
DB_SCALE_SLIDER() Calculates the slider position, 0 - 255, for a given `sDB_LISTBOX` structure.	The `DB_SCALE_SLIDER` copies the slider value to the `sTempListBox` structure. Syntax: `DB_SCALE_SLIDER(_sDB_LISTBOX sTempListBox)` Variables: sTempListBox = Represents the list box structure. (Required.) Return Values: `DB_SCALE_SLIDER` returns a number representing the position in the list, with a value of 0 - 255. Value is calculated for touch panels from the front of the list return 255 (top of a slider).
DB_STRING_ REPLACE() Converts a string to a URL compatible string. All characters illegal in URL's are replaced with a "%" hex code equivalent.	The `DB_STRING_REPLACE` function is used to convert SQL queries into CGI strings. Syntax: `DB_MAKE_URL_STRING(CHAR strSTR[])` Variable: strSTR = Represents the string to have search/replace operation applied to. (Required.) strSEARCH = Represents the search target. (Required.) strREPLACE = Represents the replace target. (Required.) Return Values: `DB_GET_XML_VALUE` returns the string `strSTR` after the search/replace operation has been applied.
DB_TRUE_FALSE() Converts the `CHAR` values of 1 and 0 into the strings "`TRUE`" and "`FALSE`", respectively.	The `DB_TRUE_FALSE` function is used to convert Boolean parameters from NetLinx into strings for XML. Syntax: `DB_TRUE_FALSE (CHAR bFLAG)` Variable: BFLAG = Represents the CHAR to be converted. (Required.) Return Values: • '`TRUE`' if the flag was non-zero. • '`FALSE`' if the flag was zero.

Putting It All Together

Once you have created all the files, it is time to put them to work. The first thing you want to do is put the database, the NetLinxDBInclude.asp file, and database server script in a path accessible by your web server. If you accepted the defaults when installing your web server, the root of the server most likely exists in **C:\Inetpub\wwwroot**. The best thing to do is create a directory under this path and place the database (MDB file) and database server script (ASP file) in this directory. If you have your web server running, you should be able to call up the file in your browser. Try typing this into your browser's address window: **http://192.168.012.175/DB/CDExample.asp**. This assumes your IP address is **192.168.12.175** and you placed your file in a directory called DB (**C:\Inetpub\wwwroot\DB**). Adjust these values to match your setup. If all goes well, you will see the following message:

```
<?xml version="1.0" ?>
- <rsUnknown>
- <scriptError>
    <errorNumber>0</errorNumber>
    <errorDescription>No SQL Supplied</errorDescription>
  </scriptError>
</rsUnknown>
```

If you do not get this message in your browser, there is no need to continue; NetLinx will not be able to access the database. There is a problem in the setup. If the file is not found, the path is likely to be wrong. If the script cannot connect to the database, the MDB file is not located in the same directory as the script or the DSN entry is improperly setup.

You may see the No SQL Supplied message without any other formatting. If you do, you are running a browser that does not support XML, but that is OK. You do not need an XML compliant browser, and NetLinx will be able to access the database.

If you get a **500 Internal Server Error** message when trying to view this page in a web browser, the web server has not installed properly. You must correct this problem before proceeding.

Once you can correctly view the database server script in your browser, double check the path setting in your call to **DB_INIT_CLIENT**. Enter the following as the IP address (or server name): **http://192.168.12.175**. The IP port number is usually **80**. In this case, you should enter **/DB/CDExample.asp** as the path to the database server script.

Now you are ready to compile and download your AXS file. Make sure the NetLinxDBInclude.axi file is in the same directory as your AXS (and maybe AXI) files. Once you have compiled, downloaded and rebooted your Master, you should be able to access the database. If you are having any problems, check the NetLinx terminal using NetLinx Studio or Telnet and watch for error messages.

Running DBWizard

File Menu

You can create, open and save DBWizard (DBW) files. The DBW file contains the database connection, all queries you have generated and all the information about the webserver connection and NetLinx code parameters. You can save your work in DBWizard; if you have a small change to make, you can add it to your existing database queries.

You should note that DBwizard will re-generate the AXI file listed in the "NetLinx AXI File" text box. Any changes you made in NetLinx Studio will be overwritten by DBWizard. You should avoid customizing the AXI file generated by DBWizard by including the AXI into and AXS programming and doing your customization in the AXS file.

The DBWizard file contains absolute paths for the database, APS and AXI files. If you move the project, you will be prompted to look for the database. Use the provided dialog to browse to the database.

The normal File menu:

- New
- Open
- Save
- Save As
- Exit

Database Tab

Under the Database tab, select the database you intend to connect to. You can either browse and open a Microsoft Access database (.MDB file), or open a Data Source Name connection. Data Source Names can connect to any ODBC compliant database, as long as you have the ODBC driver installed for the database. You will also need to have the ODBC driver installed on the PC running the web server.

Select a database and click **Open**. Once a database is opened, you can build your queries and generate NetLinx code.

When connecting to an ODBC database via the Data Source Name connection, if the database requires a username and password authentication, you can add the username and password in the appropriate fields at the bottom of the DSN tab.

Queries Tab

The Queries tab is where you build and decide how to access the database from NetLinx. On the left, you will see a set of tabs for Tables and Views. This represents the information contained in the database.

When you select a table or view, all the fields for it will show up in the "Fields For..." window. For a given table or view, you can select any or all fields to read or write to the database. In Microsoft Access, any query entered under the Queries tab will show up in DBWizard under the View tab.

Once you have picked a set of fields, enter a query name in the upper right text box. This name will be used to identify which set of information you are talking about in the database. Use a meaningful name here: for instance, if you are reading the name of CD's and their artists, call the query CDNames or something that makes sense to you.

When you have filled in the name, choose whether you want to read, add, update or delete this information in the database. If you have chosen a view, only read is allowed. When you choose read, the **Page Size** box is enabled. Enter how many records you want to see at one time on your touch panel here.

Once you have picked your access methods, click **Save**. When query name is moved to the Queries box in the bottom right, you are ready to build your next query. You can enter multiple queries for any table or view in the database if you want to get different information or have the information sorted in a different way.

For existing queries, you can review the properties with the **Properties** button. You cannot edit the information here so if you make a mistake, delete the query using **Delete** and add it again.

If you want records sorted, choose **Sorting** and add sorting information to any query.

Sorting

When you press the **Sorting** button for a query, you will be presented with the Sorting dialog. All the fields for the query will appear in the left hand list box. To sort by a given field, select the field and hit the **->** key or double click the field. The field will be moved to the **Sorted Fields** list box. The **Ascending** and **Descending** buttons at the bottom of the **Fields** list box allow you to determine if you want the fields sorted in ascending or descending order.

Once a field(s) is in the Sorted Fields list box, you can change the sorting order by selecting the field and pressing the up or down buttons at the bottom of the list box.

If you make a mistake and do not want to sort a field anymore, select the fields and hit the **<-** button or double click the field. The field will then be moved to the "Fields" list box.

When you are finished, click **OK**. If you do not want to save your sorting changes, click **Cancel**.

File Tab

The options under the File tab allow you to control the NetLinx code (AXI file) and database server script (ASP file) that will be generated. The options are:

- ASP File
- Webserver Host name or IP Address
- Webserver IP Port
- Webserver Path
- Absolute Database Path
- NetLinx AXI File
- NetLinx Local IP Port
- NetLinx Code Prefix
- Encapsulate

Once all these values are populated, you can select **Generate Code** from the Tools menu to generate the code. Four new files will be created:

- The automatically generated ASP and AXI files
- NetLinxDBInclude.ASP and NetLinxDBInclude.AXI

These files include the helper function that the generated ASP and AXI files rely on.

ASP file

This is the main database server script file and will be transferred to the web server when you are finished.

If this file exists, you will be prompted to overwrite it.

Webserver host name or IP address

The address of the web server that is running the ASP file.

The NetLinx code will require this value in order to work. If you do not know the name or address of the web server, you can use a place holder like "localhost" and change the AXI manually in NetLinx Studio at a later time.

- If you have access to the server, you can use the **winipcfg** command under Windows 95/98 or **ipconfig** under Windows NT/2000 to get the IP address of the server.
- If you change any of these parameters, a sample browser URL will be built for you in the "Example Browser Path" window.

Webserver IP port

The Webserver IP port is a web server port, usually port **80**. Only unusual web server setups will use another value. If 80 does not appear to work, contact your web server administrator for the correct value.

If you change any of these parameters, a sample browser URL will be built for you in the "Example Browser Path" window.

Webserver path

The path from the web server root where the ASP file will exist. This is the path relative to the web server root where the database and database server script will be placed.

If you do not know where this will be yet, leave it blank. If you have control of the web server, enter a simple directory name that represents your application. Then you can create this directory on your web server and place the database and database server script in this directory.

As you enter this value, a sample browser line will be built for you in the "Example Browser Path" window. You can use this example browser path to test you web server setup, as described in the *Putting It All Together* section on page 28.

Absolute database path

This will include the full path to the database in the database server script (ASP file). If you are running DBWizard on the server machine, you can check this box and you will not need to move the database. If it is unchecked, the database and database server script (ASP file) will need to be in the same directory on the web server. This option does not apply to DSN databases.

NetLinx AXI file

This is the NetLinx database interface file and needs to be included in your program. If this file exists, you will be prompted to overwrite it.

You should note that DBWizard will re-generate the AXI file listed in the NetLinx AXI File text box. Any changes you made in NetLinx Studio will be overwritten by DBWizard. You should avoid customizing the AXI file generated by DBWizard by including the AXI into and AXS programming and doing your customization in the AXS file.

NetLinx local IP port

This is the local port number to be used in the IP device definition. This value must be 2 or greater, and must not conflict with any existing local port numbers used in other IP device definitions in your program.

Local ports for IP device definitions should be sequential and not skip large blocks of numbers.

NetLinx code prefix

This prefix will be added to all variables created in the AXI file. This allows you to include multiple AXI files generated by DBWizard into the same program by giving each file a unique code prefix.

Encapsulate

The "Encapsulate" checkbox option will force DBWizard to add square brackets around each database field name during the code generation process.

(i.e. `[FieldName]`)

This option should only be used if you are connecting to a Microsoft Access Database or Microsoft SQL Server. The square brackets are used by Microsoft Access and Microsoft SQL Server to help identify table and field names in the SQL syntax.

This option's setting is saved with the DBWizard project file (*.DBW).

Writing Your AXS File

Now that DBWizard has generated the AXI file for you, you still have some work to do.

The AXI generated by DBWizard now contains all the code necessary to read and write information to the database, as well as the basic infrastructure required to send your requests to the web server. However, you still need to decide when and what you need to write to and from the database.

AMX reserves the right to alter specifications without notice at any time.

brussels • dallas • los angeles • mexico city • philadelphia • shanghai • singapore • tampa • toronto* • york
3000 research drive, richardson, TX 75082 USA • 469.624.8000 • 800.222.0193 • fax 469.624.7153 • technical support 800.932.6993