

MaxBasic

Tutorial and User's Guide Basic Programming for Advance Maxum

.....	Fehler! Textmarke nicht definiert.
Introduction and Prerequisites	4
Installing MaxBasic.....	4
Development Environment.....	5
Writing a MaxBasic program	5
Opening the .bas file.....	5
Editing	7
Saving the .bas file	7
Testing on the Workstation.....	7
Connecting to the database	8
Setting Input Parameters.....	8
Running the Program.....	9
Correcting the Program	10
Downloading and Testing on the Maxum	10
Working with the Program Table	11
Scheduling MaxBasic Programs to Run	12
Frequency/Time of Day.....	12
Event.....	12
MaxBasic programming	12
Declaring Variables	12
Variable names	13
Variable Scope.....	13
Data types	13
Arrays	13
Program Control	13
Do...Loop.....	13
For...Next.....	13
If..Then...elseif...else...endif	14
Select Case	14
Extracting Information from the Database	14
Table	14
Snapshot	14
Dynaset.....	14
Recordset Navigation	15
MoveFirst, MoveLast, MoveNext, MovePrevious	15
FindFirst, FindLast, FindNext, FindPrevious	15
Changing information in the Database	16
Setting Database Attributes from MaxBasic	16
Using Table and Dynaset Objects.....	16
Using SQL Update	17
MaxBasic Quick Reference	18
SQL Quick Reference.....	21
Primary Keys.....	21
Data Types.....	21
SQL Built-in Functions	21
Conditions	22
Select Statement	22
Update Statement	23
Send Statement	23
Some Database Tables.....	24
APPLICATION.....	24
STREAM.....	25
PROGRAM	26

PROGRAM_SCHEDULE	27
APPAO	27
APPDO	28
APPAI	29
APPDI	30
STREAM_METHOD	30
ALARMHANDLER.....	31
RESULT	32
EXTRESULT	33
PARAMETER.....	34
Class Exercises	35

Introduction and Prerequisites

The purpose of this document is to provide users with the tools necessary to develop Basic programs for the Advance Maxum Gas Chromatograph. Programs are necessary to duplicate programs that exist for the Advance Optichrom or to develop new functionality for the Advance Maxum. It is critical to have a detailed knowledge of certain Maxum tables, in order to develop MaxBasic programs. Many functions that required Basic programs on the Optichrom do not require a Basic program on the Maxum. These built-in Maxum functions can only be ascertained by knowing the Maxum tables and how they interact. A Maxum Database 3.0 document should be consulted for details on these tables, as a minimum:

**APPDI,
APPDO,
APPAI,
APPAO,
PARAMETER,
APPLICATION,
RESULT,
STREAM_METHOD,
PROGRAM,
PROGRAM_SCHEDULE**

These definitions are included at the end of this document for your convenience. Other tables may be necessary for complex tasks.

MaxBasic provides a subset of Visual Basic 3.0. A Microsoft Visual Basic 3.0 Programmer's Guide and Language Reference can be helpful, but confusing, since commands related to visual objects are not supported. The MaxBasic Quick Reference section should be used as an initial resource, while specific syntax and arguments are available in the Microsoft documentation.

Installing MaxBasic

MaxBasic is both a PC-based development environment/Basic compiler and interpreter and a Syscon-based Basic interpreter. Normally, a program is written on the PC and later attached to a Maxum application for final implementation. This involves two levels of testing, since the Basic interpreter on the PC is limited to reading values from the database. Also, The Basic interpreter on the Syscon is slightly different, requiring final testing there.

Create a MaxBasic(or some other name you like) directory on the PC. Extract all the files from the .zip file into that directory. The files are available under P18Demo\MaxBasic. Create a subdirectory for programs. Copy the file **aa1.bas** from the MaxBasic directory to the programs directory. Copy the **MaxBaxic.exe** to your desktop. To start the development environment, double-click on the MaxBasic icon.

Development Environment

The MaxBasic development environment allows you to create, edit, compile, run on the PC, create and attach code to a Maxum program, and edit an existing Maxum program. To start MaxBasic, double-click the icon:

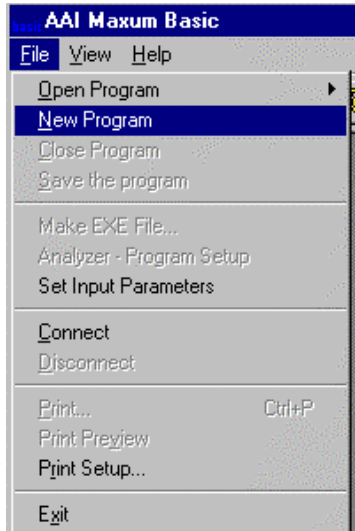


Writing a MaxBasic program

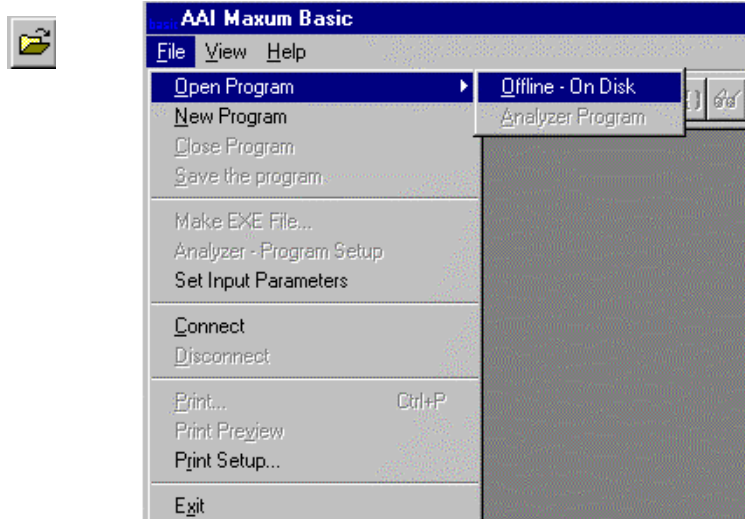
A basic program exists in two formats: source code and pseudo code. The source code(scodel) is the form that you can view and edit. It is the only form needed to compile and interpret on the workstation. It resides in the .bas file. The pseudo code(pcode) is compiled from the .bas file into the .exe file. Both files are attached as binary attributes to the program table during the setup of the program on a Maxum.

Opening the .bas file

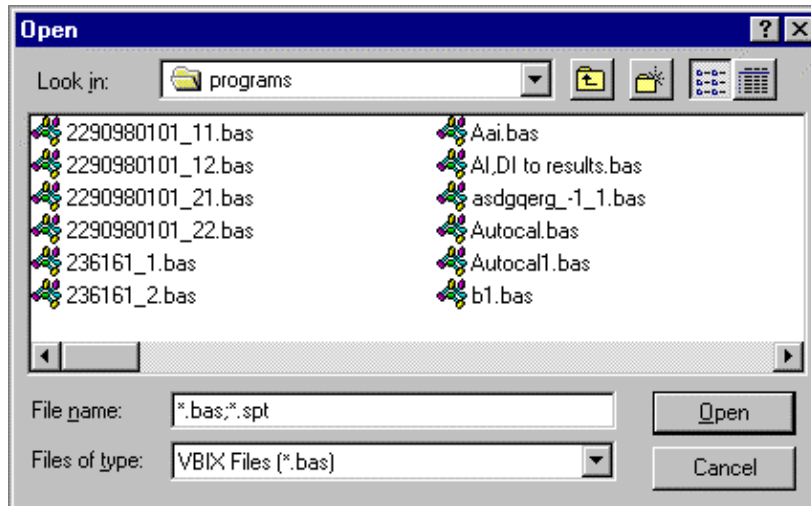
Use this for opening a new basic program:



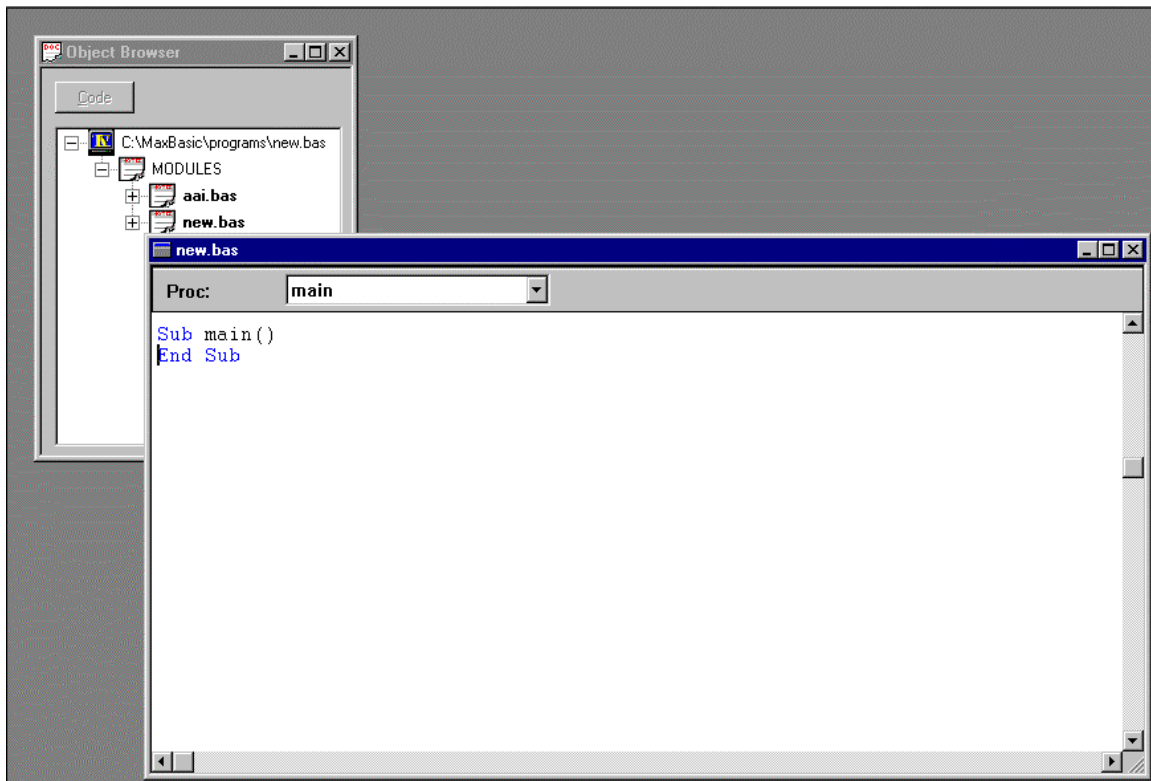
Use one of these to open an existing basic file for editing:



Select the program from the list box or key in a new file name:



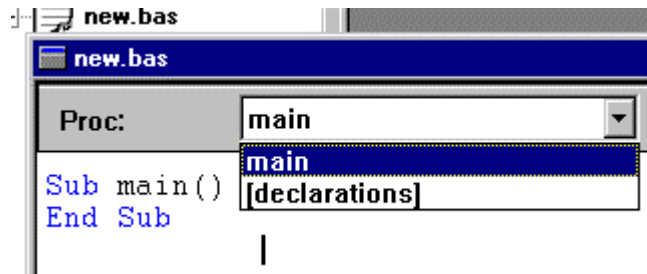
These two windows will appear:



The object browser window shows two modules:

Aai.bas contains the program's passed arguments(application, stream, program id, iargs and rargs)
New.bas is the window displayed on top of the Object Browser. In this case, I created a new program called **new.bas**

The drop down box at the top of the new.bas window gives a list of sub procedures and global declarations that are contained in your program. You may use this box to navigate to the different parts of the program.



The general form of a MaxBasic program is:

```
Sub main()  
    (Program lines)  
End Sub
```

Editing

You will notice that the Basic keywords, like **Sub** and **End** are displayed in blue. This can be helpful, but in this version of the development environment, it is not consistent.

Use the cut, copy, paste edits very cautiously. You will notice that all that you copy may not appear or additional weird characters will appear.

Saving the .bas file

Use



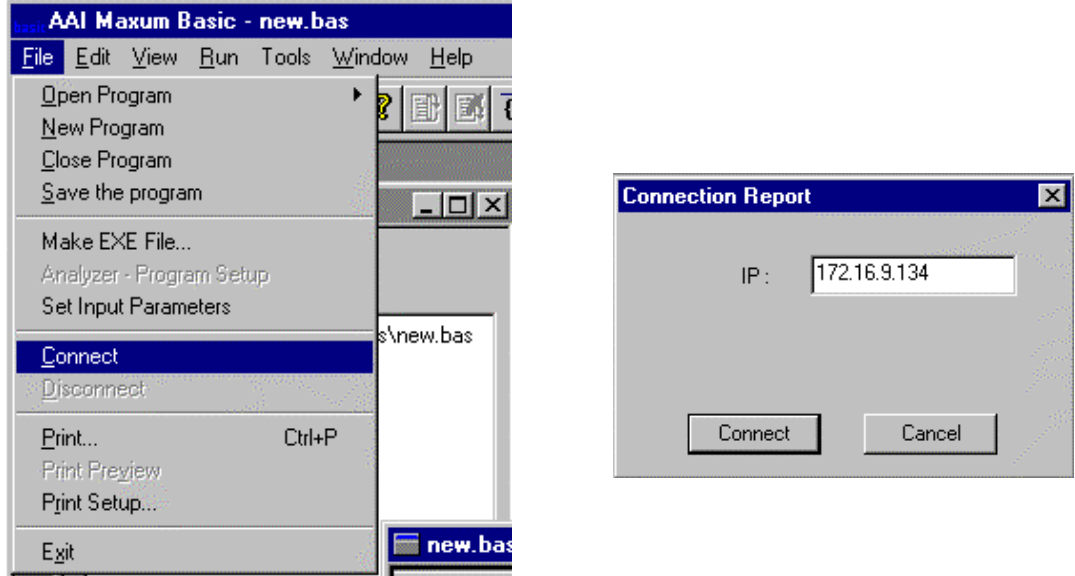
to save the .bas file also know as scode.

Testing on the Workstation

Testing a MaxBasic program on the workstation allows you to correct compile errors and check to see if the data is extracted from the database as you expected. Final testing on the Maxum is always required, since slight differences have been encountered in the Basic interpreters that could cause your program to execute differently on the Maxum.

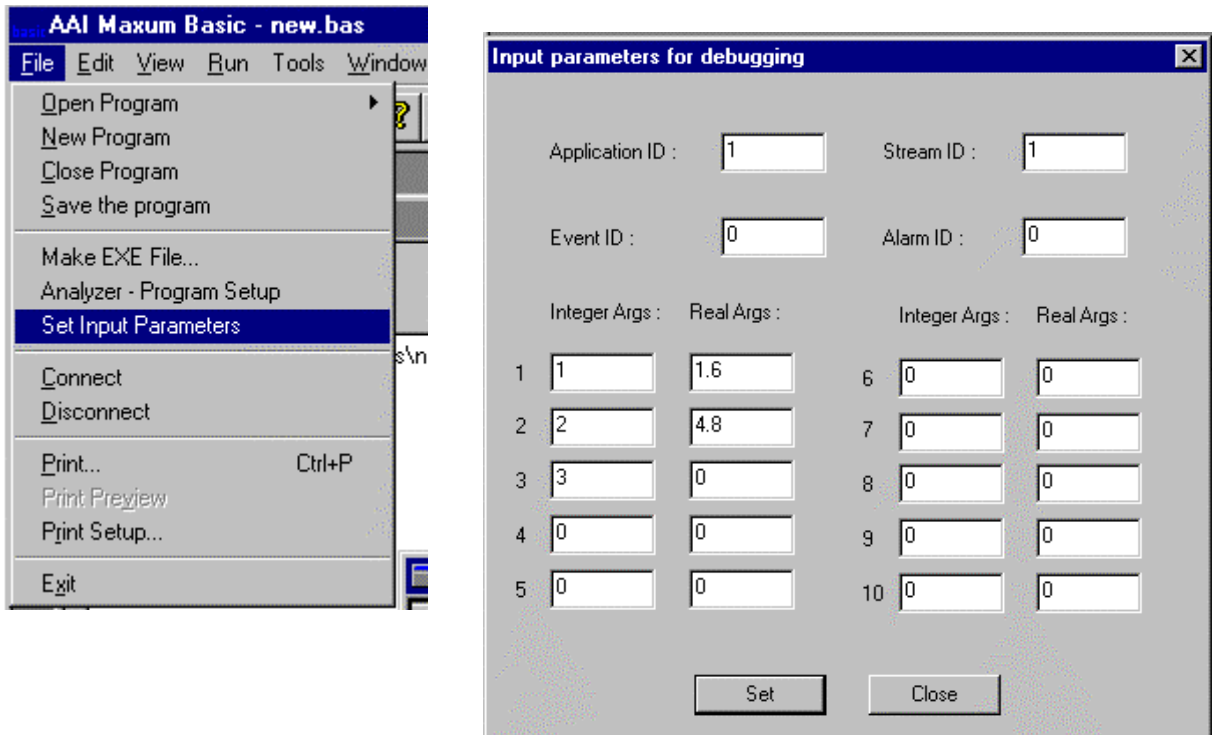
Connecting to the database

In order to test extraction of information from the database, it is necessary to attach to a Maxum for testing. Use the IP address for connection to the Maxum:



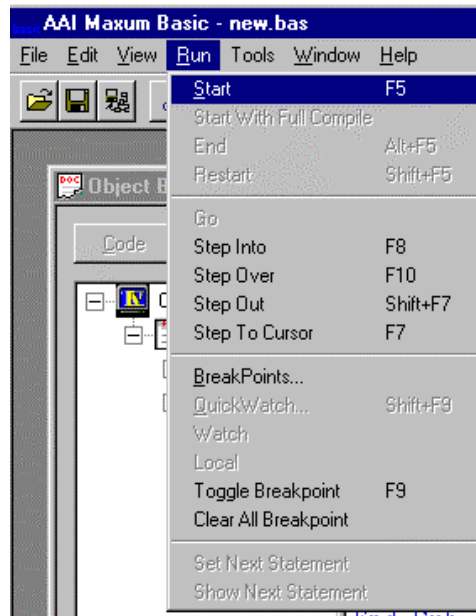
Setting Input Parameters

Most MaxBasic programs will require knowledge of the application and stream and program arguments (iargs and rargs). These are sent to the program at run time by the cycle_event, MMI, or entries in the program table. To test a program on the PC, you will need to furnish these items:

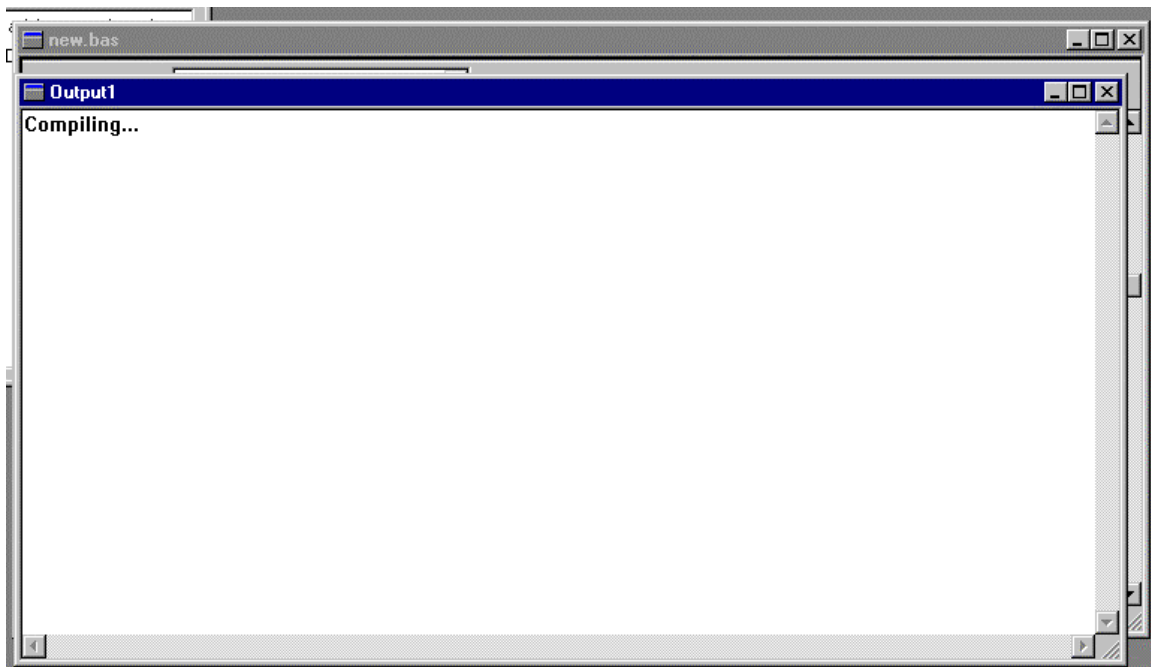


Running the Program

To run the program on the PC:



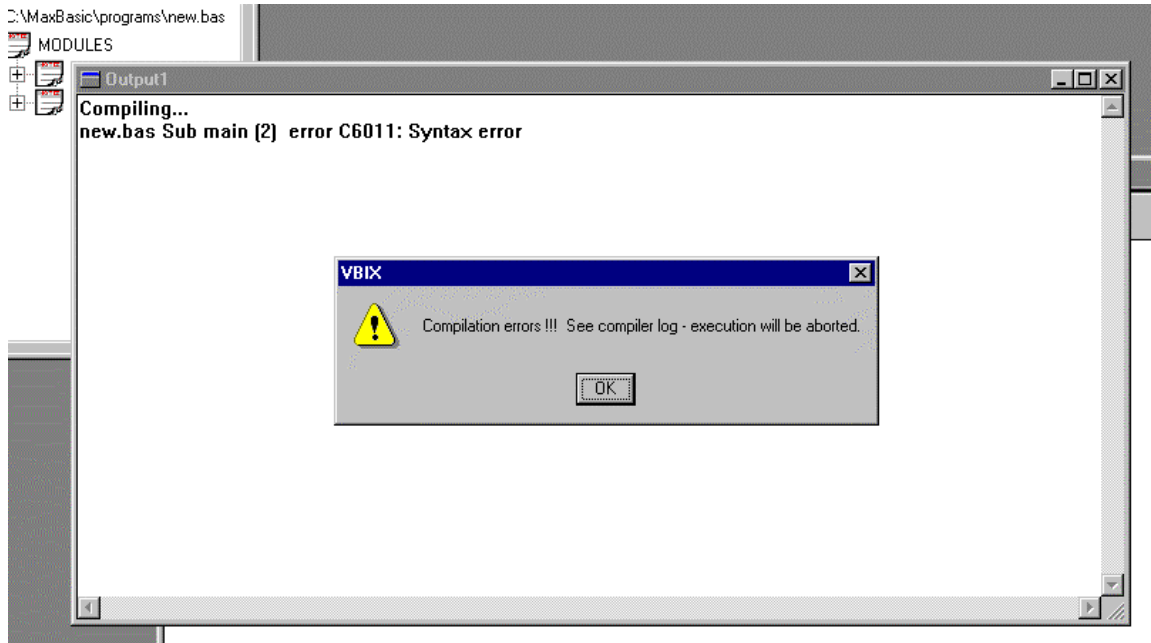
If the program compiles and runs, you will see:



Unless you have used the command MsgBox, this is all you see. There is no indication of what database information was extracted or what the program would have done.

Correcting the Program

If the program doesn't compile, you will see something like this:

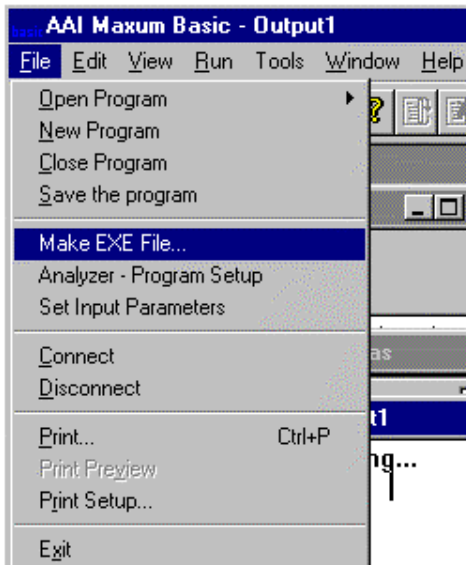


In this case, there is a syntax(grammar) error in line 2. There are no visible line numbers in the program, so it is necessary to count down to the line of interest. There are many errors that can occur. The method for correcting these errors is:

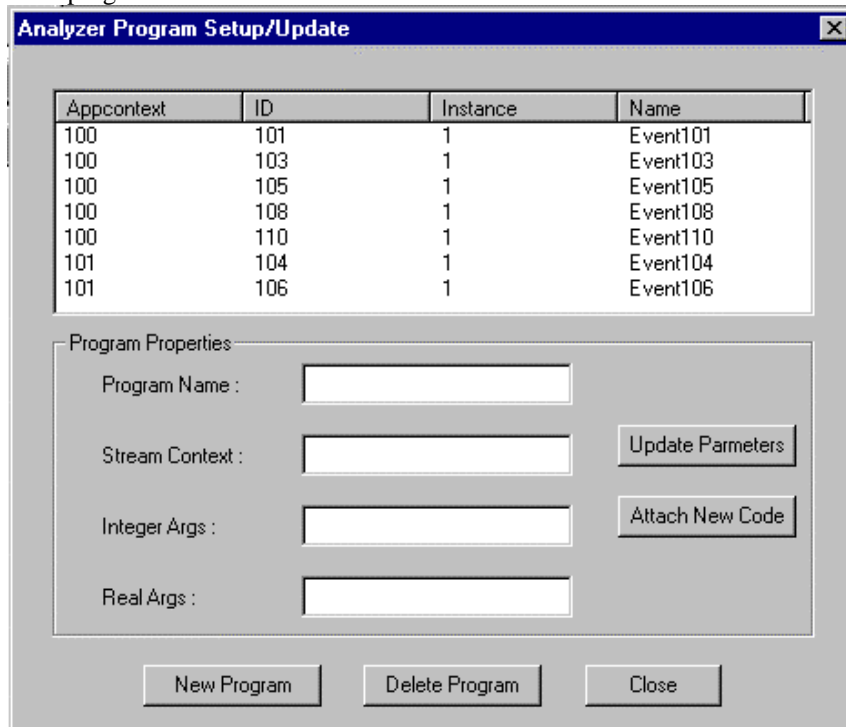
- ✓ go down the list sequentially
- ✓ go to the line number in the .bas file and try to figure out what is wrong
- ✓ correct one item at a time and recompile, since many errors cause a domino effect
- ✓ if you can't figure out what is wrong, contact your MaxBasic support representative

Downloading and Testing on the Maxum

Once the program is compiled and runs, downloading to the Maxum first requires an .exe file, also known as pcode:



To attach the program to the Maxum:



Create a new program by using the **New Program** button. To attach the .exe to an existing program, use **Attach New Code**.

Working with the Program Table

The attributes of the Program table are:

Appcontext – designate the application for the program or -1 for all applications

Id – event id

Instance – integer that makes an occurrence of a program within application unique. This allows the same event to be in an application twice with different code or arguments for different streamcontexts.

Streamcontext – comma separated list of stream ids or -1 for all streams (this attribute is checked for validity when the program runs)

Name – Unique name for this event. It is enforced to be the same for all instances of this event within an application.

Iargs – comma separated list of parameter ids(from the parameter table) which are passed as integer arguments to the program at run time. These parameters are evaluated at the program run time and remain static during run.

Rargs – comma separated list of parameter ids(from the parameter table) which are passed as float arguments to the program at run time. These parameters are evaluated at the program run time and remain static during run.

Runstream – This attribute is used only when the RUN button is used on the MMI. Therefore, it is only useful in testing a program that will run as a cycle event, by furnishing a stream.

Pcode – The .exe file (binary)

Scode – the .bas file (binary)

Status – Run(-1), Running(-2), Success(0), Failed(1), Cancel(2)

Max_exec_time – Not used in Version 3.0. Intended to be used as a governor for the Basic task.

Enable – enabled for frequency or time of day scheduling

Pgmfunction – autocalibrate(1)

The primary key to the program table is **Appcontext, Id, Instance**. These are unique and can't be modified once created. The rest of the attributes can be modified.

Scheduling MaxBasic Programs to Run

Once a program has been loaded into a Maxum analyzer, it can be run by two main methods:

Frequency/Time of Day

Programs that are scheduled to run by frequency or time of day are attached to the program table and the scheduling information is added on the MMI or System Manager. It will run without a stream argument passed to it, so references to the passed argument **Strm** will yield 0. The passed argument **Appl** will contain the appcontext. **Iargs** and **Rargs** must be in the parameter table with streamcontext -1. The streamcontext in the program table is ignored.

Time of Day scheduling:

Enter time in attribute **Schedule_time** (examples: 10:00, 13.00, 9)

Enter day of week mask in **Schedule_day**: 127= 1111111 – every day

21=10101 – T Th Sa

42=101010 – M W F

Or

Enter day of month + 200 in **Schedule_day** (examples: 201, 216, 220)

Frequency Scheduling:

Enter units in **Freq_Unit**(1=hour, 2=minute, 3=day, 4=sec, 5=ms, 6=month)

Enter amount in **Freq_amt**

The program must be enabled for scheduling to start. Frequency programs will start from the time the program is enabled. To test the code, Go to **Menu/Setup/Basic Events**. Select the program and push **RUN**. Pushing **RUN** ignores the enable/disable state of the program.

Event

Programs that will run from an event can be run from a **cycle_event** within a method, An external Advance DataHiway command, a **limit/alarmhandler**, or from a **mvrprogram** in the stream_method table:

- Attach the program to a cycle event in EZChrom. After the program is added into the program table, reconfigure the instrument. Select the program as an event to run at a particular cycle time. Enable/disable state is ignored when running a cycle event
- Activate from an HAE (Advance DataHiway) message. Stream is furnished by the HAE message. The program must have appcontext of the default ADH application(system_control table). Enable/disable state is ignored.
- Attach a program to the **alarmhandler** table to be activated by a limit. The program will run with the Application_id of the Alarmhandler and the current stream_id of the application. Enable/disable state is ignored.
- Attach a program to the **mvrprogram** of stream_method(runs at the end of cycle, if no errors). It will run with the application and stream of the stream_method table. Enable/disable state is ignored.

Test both program types by going to **Menu/Setup/Basic Events**. Select the program and push **RUN**. If the program requires a stream, enter a **runstream** from the **Detail** screen. Pushing **RUN** ignores the enable/disable state of the program.

MaxBasic programming

Declaring Variables

Variables are declared with the **DIM** statement. MaxBasic does not require a variable to be declared, but it is good programming practice to explicitly declare you variables.

Variable names

Variable names must start with a letter, must contain only letters, numbers and the underscore character(_), must not exceed 40 characters, and cannot be a reserved word.

Variable Scope

A variable is declared local (with **Dim**) or **Global**. A global variable is available in all procedures in your module. A local variable remains in existence only as long as the procedure is executing.

Data types

The **as** clause allows you to explicitly set the data type of the variable. If a variable is not explicitly declared with an **as** clause, it defaults to data type **Variant**. You can store any type of data in a **variant**, but the use of **variants** is discouraged. Other data types are: **integer**(16-bit), **long**(32-bit), **single**(32-bit float), **double**(64-bit float), **string**. Objects (**table**, **dynaset**, **snapshot**) are also declared. Caution: Since MaxBasic will default your variables to **Variant**, slight misspelling of variable names within your code will cause unexpected results. For example:

```
Dim MyTable as Table
. . .
MyTabl = db.opentable("select anlz_id from analyzer")
```

Will result in a strange compiler error on the opentable statement.

Arrays

Arrays are collections of data of the same data type. They can be single or multi-dimensional:

```
Dim Num(10) as double
Global myints(3,3) as integer
```

Noteworthy bugs in 3.0: all your arrays need to be declared global.

Program Control

Do...Loop

The Do loop executes a block of code an indefinite number of times, based on the evaluation of a true/false condition. It takes several forms:

- **Do Until** x = y
....
Loop
- **Do While** x = 1
....
Loop
- **Do**
....
Loop While Not MyTable.EOF

You can optionally exit a Do loop prematurely with an Exit Do statement.

For...Next

The For loop executes a block of code a definite set of times:

- **For** I = 1 **to** n
....
Next I
- **For** j = 10 **to** 1 **step** -1
....

Next j

You can optional exit a For loop prematurely with an **Exit For** statement.

If..Then...elseif....else....endif

The If Then bases execution of blocks of code on one or more conditions:

- **If x = y then** y = z
- **If x = y then**
 a = b
 else
 a = c
 endif
- **If x = 1 then**

 elseif x = 2 then

 Else

 endif

Select Case

The Select Case structure executes blocks of code selectively.

- **Select x**
 Case 1
 ...
 Case 2, 4, 6, 9 to 10, is < 0

 Case Else

 End Select

Extracting Information from the Database

Maxbasic allows three recordset objects to extract and modify information from a database table:

Table

A table is a type of recordset that represents a view of a database table or joined tables. The data in a table object are static. The records in a table object can be acted upon by the Edit and Update methods. If the records are intended for read-only, a better object to use is a Snapshot.

```
Dim MyTab as Table  
MyTab = db.OpenTable("Select application_id,id,value from appdi where application_id=2")
```

Snapshot

A snapshot is a recordset that is intended for read-only.

```
Dim MySnap as Snapshot  
MySnap = db.CreateSnapshot("select application_id, stream_id, result_index from result where application_id=4")
```

Dynaset

A Dynaset is a recordset that is not only editable, but its contents are dynamic.

```
Dim MySet as Dynaset  
MySet = db.CreateDynaset("select anlz_id from analyzer")
```

Recordset Navigation

To navigate through a recordset for the purpose of extracting information from individual records or modifying individual records use one of these sets of methods:

MoveFirst, MoveLast, MoveNext, MovePrevious

This set of methods moves through the recordset in the order that it was extracted(or ordered):

```
MyTable.MoveFirst  
Do Until MyTable.EOF  
    MyTable.MoveNext  
Loop
```

FindFirst, FindLast, FindNext, FindPrevious

This set of methods moves through the recordset based on a where clause:

```
MyTable.FindFirst("result_index > 1")  
Do Until MyTable.NoMatch  
    MyTable.FindNext("result_index > 1")  
Loop
```

Changing information in the Database

Setting Database Attributes from MaxBasic

Noteworthy bugs in 3.0:

1. **float attributes must never be set to integer values (even if they are in float format(don't use 1 or 1.000). One temporary solution is to add .00001 to every float you set.**
2. **Any null value in the database cannot be set with the edit/update methods.**

- **Database Bool**

Declare string variables to contain the strings "TRUE" and "FALSE." Use these variable to set the attributes:

```
Dim tr as string
Dim fl as string
Tr = "TRUE"
Fl = "FALSE"
....
If x =1 then
    MyTable("enable") = tr
Else
    MyTable("enable) = fl
endif
```

- **Database Integer**

Use an integer variable or a constant :

```
Dim myint as integer
Myint = 2
....
MyTable("trtval") = myint
MyTable("logval") = 1
```

- **Database Float**

Use a single or double variable or constant. Make sure value is not an integer:

```
Dim myfloat as double
Myfloat = 3.45
....
MyTable("saved_value") = myfloat+.00001
MyTable("Buffered_Value") = 0.00+.00001
```

- **Database Datetime**

Datetimes are set from strings(must be of format dd-mmm-yyyy hh:mm:ss):

```
Dim mystring as string
Mystring = "13-SEP-1999 13:00:00"
....
MyTable("saved_time") = mystring
```

Using Table and Dynaset Objects

To modify a single record in a recordset, use the edit, update methods:

- **Dim MyTable as table**

```
MyTable = db.opentable("select application_id,id,value from appai where application_id=1 order by id")
```

```
MyTable.MoveFirst
```

```
Do Until Mytable.EOF
```

```
    If MyTable(2) = 1000 then
```

```
        MyTable.Edit
```

```
        MyTable(3) = 60.0
```

```
MyTable.Update
Elseif MyTable("id") = 1001 then

MyTable.edit
MyTable("value") = 40.0
MyTable.Update
Endif
MyTable.MoveNext
Loop
```

←Note that you can refer to an attribute by name or by order in the recordset (first one being zero)

Using SQL Update

By using the **ExecuteSQL** method, you can modify one, many, or all records in a database table with a single statement. Another advantage in using the SQL update statement is that the records don't have to be first extracted from the database:

To duplicate the Table update code listed above:

```
Dim numrows as long
```

```
Numrows = db.ExecuteSQL "Update appai set value = 60.0 where application_id = 1 and id = 1000"
```

```
Numrows = db.ExecuteSQL "Update appai set value = 40.0 where application_id = 1 and id = 1001"
```

Caution: These update statements are so powerful, that entire database tables can be easily destroyed. Other SQL statements are supported, like delete or insert, but these are extremely dangerous. Be sure that a backup copy of the database is made.

MaxBasic Quick Reference

Basic	Type	Usage: example
Call	Statement	Calls another sub procedure: Call findparams
Cdbl,csng,cint,cstr	Functions	Converts data type: Myint = cint (mystring)
CreateDynaset	Method	Creates a set of data that changes dynamically: Myset = db. createDynaset ("select anz_id from analyzer")
CreateSnapshot	Method	Creates a snapshot of data from the database: Mysnap=db. createSnapshot ("select anz_id from analyzer")
Date	Function	Returns date (\$ indicates string, else variant): Msgbox date \$
Dim	Statement	Declares variables: Dim I as integer Dim timestamp as string Dim Res as table Dim myvar as variant
Do Until.....Loop	statement	Executes a section of code until a condition is true: Do until Res.eof Res.movenext Loop
Do While.....loop	Statement	Executes a section of code while a condition is true Do while not Res.eof Res.movenext Loop
Dynaset	Object	A set of database records that keeps the data "fresh", i.e. , reflects changes in the data since the program started.
Edit	Method	Prepares current record for editing: Res. edit
End	Statement	Ends procedure or block: End sub End if
eof	Property	Use after moverfirst or movenext to see if at the end of the recordset: If Res.EOF then exit sub
ExecuteSQL	Method	Executes an Update or Send SQL statement: Numrows = db. ExecuteSQL "update result set saved_time = cycle_runtime where application_id=1"
Exit do	Statement	Interrupt and exit a do loop: Do while not Res.eof If Res(1) = 1 then exit do Res.movenext Loop
Exit for	Statement	Interrupt and exit a for loop: For I = 1 to 15 I=I+1 If num(I) = 0 then exit for Next I
Exit sub	Statement	Exit the sub procedure: If strm = 30 then exit sub
Findfirst	method	Find the first record that satisfies the where clause: Res. findfirst ("application_id=1 and stream_id=2")

Findnext	method	Find the next record that satisfies the where clause: Res. findnext ("application_id=1 and stream_id=2")
For n = a to b.....next n	Statement	Executes a section of code with a counter: For I = 1 to 15 I=I+1 Next I
Global	Statement	Causes a variable to available to all sub procedures in the .bas file: Global n(20) as integer
If....then.....else....endif	Statement	If num(2) = 4 then I=4 Else I=3 Endif
If....then.....elseif.....elseendif	Statement	If num(2) = 4 then I=4 Elseif num(2) = 3 then I=3 Else I=2 Endif
If....then.....endif	Statement	If strm = 3 then Msgbox "stream 30" Exit sub Endif
If....then.....	Statement	If strm = 30 then exit sub
Lcase Ucase	Function	Converts to lower or upper case(\$ indicates string, else variant): Set mystring = lcase\$(mystring)
movefirst	Method	Moves to the first record in a set of records: Res. movefirst
movelast	Method	Moves to the last record in a set of records: Res. movelast
movenext	Method	Moves to the next record in a set of records: Res. movenext
Msgbox	Statement	Print a message to the "screen": Msgbox "my num is " + str\$(num)
nomatch	Property	Use after findfirst or findnext: If Res.NoMatch then Exit sub endif
Now	Function	Gives the current local time: Nowtime = now
On....gobsub On....goto	Statement	"computed goto" – I is an index into a list of labels or subroutines:: On i goto dothis,dothat
opentable	Method	Extracts a record set from the database that satisfies the select statement: Res = db. opentable ("select anlz_id from analyzer")
recordcount	Property	Use after movelast method to get current recordcount in a table, dynaset, or snapshot N = Res. recordcount
Right\$, Left\$, Mid\$	Function	Get the righthand, lefthand, or middle of a string: newstring = right\$(oldstring,4) newstring = left\$(oldstring,2)

		<code>newstring = mid\$(oldstring,3,2)</code>
Rtrim Ltrim Trim	Function	Trim leading blanks from the left, right, or both(\$ returns a string, else a variant): Newstring = Trim\$(oldstring)
Select		Selectively run a block of code Select result_name Case "Hydrogen" Case "Ethylene" Case else End select
Set	Statement	Set num = 3
Snapshot	object	A set of records from the database that is read-only
Str	Function	Returns a string representation of a numeric value(\$ returns string, else variant): <code>Mystring = str\$(mynum)</code>
Sub	Statement	Declares the name, arguments for a sub procedure
Table	Object	A set of records from the database that is editable, but not dynamic.
Time	Function	Returns system time(\$ indicates string, else variant): <code>Mytime = time\$</code>
Update	Method	Saves values into a table or dynaset Res.edit <code>Res(1) = 2</code> Res.update

SQL Quick Reference

The Maxum supports a subset of ANSI standard SQL (Structured Query Language)

Primary Keys

Each database table has a primary key. It is an attribute or set of attributes that make the table record unique. When extracting information from a table, it is always best and often required to select all the primary key attributes. By definition, a where clause that calls out the record by primary key guarantees that only one record is extracted or acted upon. Note that each application does not have its own set of tables, as it may appear in System Manager. Therefore, all results for the Maxum GC are stored in a single table. This will require **application_id** to be used in all the where clauses to narrow the record set to one application, if required. The primary key for a record cannot be changed.

Data Types

The format for data in the SQL statement is dependent on the data type of the attribute in the table. You will need to refer to the database document for details on the data types of attributes. It is not sufficient to look in System Manager, since it does not carry this information. Examples for different data types:

1. Bool
 - ✓ **select id from appdi where enable = true**
 - ✓ **Update stream_method set trtnow = true where stream_id=1**
2. Integer(32)/Integer8/Integer16
 - ✓ **select application_id, id,value from appdi where application_id=1**
 - ✓ **Update result set trtval = 3 where application_id=1 and stream_id=2 and result_index = 4**
3. Float(64)/Float32
 - ✓ **select application_id, stream_id, result_index, saved_value from result where saved_value > 0.0**
 - ✓ **Update result set saved_value= 123.456 where application_id=1 and stream_id = 2 and result_index=5**
4. Datetime
 - ✓ **select application_id, stream_id, result_index, saved_value, saved_time from result where saved_time > '15-SEP-1999 10:00'**
 - ✓ **Update result set saved_time = '15-Sep-1999 13:00'**
5. Char
 - ✓ **select application_id, stream_id, stream_name from stream where stream_name = 'Stream 1'**
 - ✓ **Update alarmhandler set text = '! This is a fault' where alarmcode = 904**
6. Binary
 - ✓ **Don't read or write to these**

SQL Built-in Functions

Certain built-in functions are available in SQL. These are primarily used for Datetime manipulation:
Some Useful functions:

Where a is a datetime attribute, n is an integer, f is a character string format, s is a character string

1. Datetime **Add_months(a,n)**
2. Datetime **Add_days(a,n)**
3. Datetime **Date(s)**
4. Datetime **Datetime(s)**
5. Datetime **DatetimeF(s,f)**
6. Datetime **Days(n)**
7. Char **Get_date(a)**

8. Char **Get_time**(a)
9. Char **Get_datetimef**(a,f)
10. Char **Get_datetime**(a)
11. Datetime **Hours**(n)
12. Datetime **Last_Day**(a)
13. Datetime **Local_time**(a)
14. Datetime **Minutes**(n)
15. Datetime **Now**()
16. Datetime **Seconds**(n)
17. Datetime **Standard_time**(a)
18. Integer **To_days**(a), integer **To_minutes**(a), integer **To_hours**(a)
19. Datetime **Today**()

Examples:

- ✓ **Update** result set saved_time = **now**()
- ✓ **Update** result set saved_time = **now**() + **Hours**(2)
- ✓ **Update** result set result_name = **Get_time**(cycle_runtime) → This is valid, but doesn't make much sense

Conditions

The Where clause on the select, send, and update SQL statements uses conditions with these comparison operators:

=	equal
<>	not equal
<	less than
>	greater than
<=	less that or equal
>=	greater that equal

conditional formats are:

condition
condition AND condition ←there is no OR

where conditions are:

expr = expr
expr <> expr
expr < expr
expr > expr
expr >= expr
expr <= expr
expr is {not} null
expr is [not] in (expr,expr,...)

Select Statement

The select statement is used to retrieve records from a database table for **tables**, **snapshots**, and **dynasets**. This statement can't be used with database method ExecuteSQL The general syntax is:

Select a, b, c, d **from** mytable **where** a = xxx **and** b =yyy **order by** a,b,c

Where a, b, c, d are attributes in the database table mytable, it must contain the entire primary key for the table. The order by clause is optional , but recommended. It may carry a direction for sorting asc or desc. When using constant values in the where clause, the data type must match the data type of the database attribute.

Examples:

- ✓ **Select** application_id, stream_id, result_index **from** result **where** application_id=1 **and** stream_id=1
- ✓ **Select** application_id, id, enable, value **from** appdo **where** application_id=100 **order by** id **desc**
- ✓ **Select** application_id, stream_id **from** stream **where** stream_name = 'my stream'

- ✓ **Select** application_id, id, value **from** appdi **where** value = true **order by** id
- ✓ **Select** application_id, stream_id, result_index, saved_value **from** result **where** saved_value < 1.00

Database joins are supported:

- ✓ **Select** x.application_id, x.stream_id, stream_name result_index **from** stream x, result y **where** x.application_id = y.application_id **and** x.stream_id = y.stream_id **and** stream_name= 'mystream' **order by** result_index

Update Statement

The update statement changes attribute values in a database table. The update will validly execute on none, some, or all records in a table. It is used exclusively with the database method ExecuteSQL. Its general syntax is:

Update mytable **set** a=xxx, b=yyy **where** d = zzz **and** c = xyz

Examples:

- ✓ **Update** stream_method **set** enable = true **where** stream_id=3
- ✓ **Update** result **set** saved_value = 0.00 **where** saved_value < 0.0 **and** application_id=1
- ✓ **Update** result **set** saved_value = buffered_value, saved_time = cycle_runtime **where** stream_id=1 **and** application_id=2
- ✓ **Update** stream_method **set** lognow = true **where** application_id=1 **and** stream_id=3 **and** method_id=1
- ✓ **Update** result **set** saved_value = buffered_value/100.0 **where** application_id=1 **and** stream_id=2

Send Statement

The send statement allows execution of a database request. Many functions are handled automatically in the database. A subset of these useful requests are listed below. The Send command will execute validly on none, some or all records in a table. It is used exclusively with the MaxBasic command ExecuteSQL. The general syntax of the send command is:

Send 'functionname' **to** mytable **where** a = xxx **and** b=yyy

If the where clause is omitted, the function will execute on every row in the table.

Function name	Table	example
autocalibrate	Application	Send 'autocalibrate' to application where application_id = 2
Manualcalibrate	Application	Send 'manualcalibrate' to application where application_id = 1
Next_stream	Application	Performs stream step: Send 'next_stream' to application where application_id = 2
Stopcalibrate	Application	Returns to previous sequence. Remains running: Send 'stopcalibrate' to application where application_id=100
Stopcalibrate_norun	Application	Returns to previous sequence. Places in hold.: Send 'stopcalibrate_norun' to application where application_id=1
Stop_stream	Application	Stops the flowing stream: Send 'stop_stream to application where application_id = 1
Enableme	Stream_method	Enables a stream: Send 'enableme' to stream_method where application_id=1 and stream_id=3
disable	Stream_method	disables a stream: Send 'disable' to stream_method where application_id=1 and stream_id=3
select	Stream_method	forces a stream: Send 'select' to stream_method where application_id=1 and stream_id=3

deselect	Stream_method	Removes force on a stream: Send 'deselect' to stream_method where application_id=1 and stream_id=3
Balance	App_detector	Balances a detector: Send 'balance' to app_detector where application_id=1 and id=3
Run	Program	Runs a program(if stream is needed, set runstream, first) Send 'run' to program where appcontext = 1 and id=41
Pollresult	Extresult	Forces a poll for extresult: Send 'pollresult' to extresult where application_id=1 and stream_id=2 and result_index=4
Setstep	Sequence_entry	Sets the next sequence entry: Send 'setstep' to sequence_entry where id = 1 and sqid=4
Setonce	Sequence_entry	Sets this step in the sequence to run once: Send 'setonce' to sequence_entry where id=1 and sqid=3
Setalways	Sequence_entry	Sets this step in the sequence to run always: Send 'setalways' to sequence_entry where id=1 and sqid=3

Some Database Tables

APPLICATION

persistent,

APPLICATION_ID	INTEGER,
application_name	char,
mode	integer,
sne_mode	integer,
active_app	integer,
active_sequence	integer,
paused_app	integer,
paused_sequence	integer,
ezchrom_reload	datetime,
curr_app	integer,
curr_seq	integer,
curr_seq_pos	integer,
next_app	integer,
next_seq	integer,
next_seq_pos	integer,
int_app	integer,
int_seq	integer,
int_seq_pos	integer,
curr_stream_id	integer,
cm_app	integer,
curr_method_id	integer,
stream_purge_clock	float32,
waitclock	float32,
total_cycle	float32,
injection_lag	float32,
clock_time	float32,
curr_error	integer,
curr_warning	integer,
curr_error_type	char,
manualcalrun	bool,
autocal	bool,

autocal_app	integer,
autocal_seq	integer,
max_cal_reps	integer,
autoclear	bool,
alarmref	array of alarm_log,
inservice	bool,
enable	bool,
primary key(application_id),	
foreign key(curr_app, curr_seq, curr_seq_pos) references sequence_entry(application_id, id, sqid),	
foreign key(next_app, next_seq, next_seq_pos) references sequence_entry(application_id, id, sqid),	
foreign key(int_app, int_seq, int_seq_pos) references sequence_entry(application_id, id, sqid),	
foreign key(cm_app, curr_method_id) references method(application_id, method_id)	

Usage: Applications are created in Advance System Manager. Most of the entries here are used by the system to run the application. Parts of this table are viewed on the MMI[Menu][Operation Mode] or the SM[System Tables][Application].

Name	Description
application_id	customer defined id
<i>application_name</i>	<i>customer defined description for application</i>
<i>Mode</i>	<i>0=hold ;1=running; 4=flow(stream purge); 6=waiting for temp or pressure; 7= set to run by setting this status, the customer can cause an application to go to run or hold state.</i>
<i>Sne_mode</i>	<i>Indicates the state of the SNE or method: 0=hold;1=run;2=load; 5 = error</i>
<i>active_app,active_sequence</i>	<i>The sequence that is active</i>
<i>paused_app,paused_sequence</i>	<i>last sequence – used for resume after calibration or validation</i>
<i>ezchrom_reload</i>	<i>Last time downloaded from EZChrom</i>
<i>Curr_app,curr_seq,curr_seq_pos</i>	<i>Current position in SEQUENCE_ENTRY table</i>
<i>Next_app,next_seq,next_seq_pos</i>	<i>Next position in active SEQUENCE_ENTRY</i>
<i>Int_app,int_seq,int_seq_pos</i>	<i>force once or always position in SEQUENCE_ENTRY</i>
<i>Curr_stream_id</i>	<i>set from SEQUENCE_ENTRY table</i>
<i>Cm_app,curr_method_id</i>	<i>Application_id in METHOD table</i>
<i>stream_purge_clock</i>	<i>countdown for purging for current flowing stream (started from STREAM table)</i>
<i>Total_cycle</i>	<i>from METHOD table(seconds)</i>
<i>injection_lag</i>	<i>from METHOD table(seconds)</i>
<i>Clock_time</i>	<i>time in cycle from SNE</i>
<i>Curr_error</i>	<i>Current cycle unacknowledged error maintained by the ALARM_LOG</i>
<i>Curr_warning</i>	<i>current cycle unacknowledged warning maintained by the ALARMLOG</i>
<i>Curr_error_type</i>	<i>current cycle severe unacknowledged error maintained by the ALARM_LOG</i>
<i>manualcalrun</i>	<i>is manual calibration running?</i>
<i>autocal</i>	<i>Is application able to do autocal?</i>
<i>Autocal_app,autocal_seq</i>	<i>Default calibration sequence</i>
<i>Max_cal_reps</i>	<i>maximum calibration replicates to save -</i>
<i>autoclear</i>	<i>automatically ackn alarms when new cycle has better results?</i>
<i>Alarmref</i>	<i>references to all alarms for this application</i>
<i>inservice</i>	<i>is application inservice? if not, there is no automvr or auto log and results are marked "not current"</i>
<i>enable</i>	<i>application enabled? This is designed to allow a limit of enabled application per customer's application license – as defined in the application_key of the system_control table. Disabled applications are forced to be in HOLD.</i>

STREAM

persistent,

APPLICATION_ID	INTEGER,
stream_id	integer,
sv_app	integer,
sv_id	integer,
chrom_refs	array of chromatogram,
stream_name	char,
calibrate	bool,
purge_time	float32,
curr_error	integer,

prev_error integer,
curr_warning integer,
prev_warning integer,
curr_error_type char,
prev_error_type char,
alarmref array of alarm_log,
clear_results bool,
recent_cyctime datetime,
bin_hdr binary,
endofcycle_flag bool,
primary key(application_id, stream_id),
foreign key (application_id) references application(application_id),
foreign key (sv_app, sv_id) references appdo(application_id, id)

Usage: Streams are configured in the System Manager. The purpose of this table is to control and monitor the stream in the application. Parts of this table are viewed on the **SM[Application][Stream]**.

Name	Description
Application_id	<i>must be a valid application</i>
Stream_id	
<i>sv_app,sv_id</i>	<i>DO that controls sample valve for this stream</i>
Chrom_refs	used internally
<i>Stream_name</i>	<i>unique name</i>
<i>Calibrate</i>	<i>is this a calibration stream?</i>
<i>Purge_time</i>	<i>time needed to purge for this stream</i>
Curr_error	Current cycle unacknowledged error – maintained by the alarm_log
Prev_error	Last cycle unacknowledged error – maintained by the alarm_log
Curr_warning	current cycle unacknowledged warning – maintained by the alarm_log
Prev_warning	Last cycle unacknowledged warning – maintained by the alarm_log
Curr_error_type	current cycle most severe unacknowledged error – maintained by the alarm_log
Prev_error_type	Last cycle most severe unacknowledged error – maintained by the alarm_log
Alarmref	– maintained by the alarm_log
Clear_results	flag for result removal
Recent_cyctime	cycle time of most recent completed cycle
Bin_hdr	from Advance EZChrom
Endofcycle_flag	used for telling when chroms can me moved

PROGRAM

persistent,
appcontext integer,
id integer,
instance integer,
streamcontext char,
runstream integer,
name char,
iargs char,
rargs char,
pcode binary,
scode binary,
max_exec_time integer,
enable bool,
status integer,
pgmfunction integer,
schedrefs array of program_schedule,
primary key(appcontext,id,instance)

Usage: Entries are created in the Advance System Manager. This table links the Program with the application. Programs can be run from the MMI, scheduled to run at a time or time interval, run from the **Alarmhandler**, and run from the Advance System Manager. This table is viewed on the **MMI[Menu][Setup][Basic Events]** or the **SM[Application] [Tables] [Program]**.

Name	Description
------	-------------

<i>appcontext</i>	<i>-1 for system events</i>
<i>id</i>	<i>customer defined</i>
<i>instance</i>	<i>used to indicate multiple instances of the same program for purposes of scheduling</i>
<i>streamcontext</i>	<i>comma delimited list of valid stream_ids, -1 meaning all streams</i>
<i>runstream</i>	<i>used to run a program manually for a specific stream</i>
<i>name</i>	<i>name that is the same for all occurrences of id in an application</i>
<i>iargs</i>	<i>comma delimited list of parameter ids from the parameter table to be passed as integers to the executing program</i>
<i>rargs</i>	<i>comma delimited list of parameter ids from the parameter table to be passed as real numbers to the executing program</i>
<i>pcode</i>	<i>Binary executable</i>
<i>scode</i>	<i>Binary source</i>
<i>max_exec_time</i> <i>enable</i>	<i>time in seconds passed to the program interpreter enabled? if disabled, normal time interval or time of day scheduling will not be active. However, the event may be run from SM, ADH command, or the MMI.</i>
<i>pgmfunction</i>	<i>0= none 1=calibrate</i>
<i>status</i>	<i>-1= ready -2 = running 0 = success 1 = failed 2 = cancelled</i>

PROGRAM_SCHEDULE

persistent,

appcontext	integer,
id	integer,
programref	integer,
sched_num	integer,
schedule_time	char,
schedule_day	Integer,
freq_amt	integer,
freq_unit	integer,
nextruntime	datetime,

primary key(appcontext,id,programref,schednum),

foreign key(appcontext,id,programref) references program(appcontext,id, instance)

Usage: Entries are created in the Advance System Manager. This table contains scheduling information for the program table. This table is viewed on the MMI[Menu][Setup][Basic Events][SCHED] or the SM[Application] [Tables] [program_schedule].

Name	Description
<i>appcontext</i>	
<i>id</i>	<i>references program table</i>
<i>programref</i>	
<i>sched_num</i>	<i>instance of schedule</i>
<i>schedule_time</i>	<i>hh:mm:ss</i>
<i>schedule_day</i>	<i>0-127 for day of week bit 0 – Sunday bit 1 – Monday bit 6 – Saturday 200-231 for day of month</i>
<i>freq_amt</i>	<i>How many hours,minutes,days,months?</i>
<i>freq_unit</i>	<i>1=hour; 2=minute; 3=day; 6= month; 7=year</i>
<i>nextruntime</i>	<i>next time scheduled to run – automatically set by enabling the event</i>

APPAO

persistent,

application_id	integer,
ID	INTEGER,
name	char,
io_status	integer,
enable	bool,
hrdwr_id	char,
unittext	char,
value	float32,
zero	float32,
fullscale	float32,

limitapp integer,
limitref integer,
hrdwrapp integer,
hrdwrref integer,

primary key(application_id,id),
foreign key(limitapp,limitref) references limit(application_id,limit_id),
foreign key(hrdwrapp,hrdwrref) references app_hardware(application_id,id),
foreign key(application_id) references application(application_id),
foreign key(hrdwr_id) references sys_ao(hrdwr_id)

Usage: Entries are created in the Advance System Manager. The application uses an AO that is defined in one of the Sys_Ao tables by selecting the Hrdwr_id. View this table on the **MMI[Menu][I/O][AO]** or the **SM[Application][Application IO][Appao]**.

Name	Description
application_id	must be a valid application
id	reference in methods for this i/o – customer defined
<i>name</i>	<i>customer defined name</i>
Io_status	set by system 0 = normal -1 = not initialized -5= overscale error -2 = general error -6= not scanning -3= comm error -7= no hrdwr_id -4= open error -8= remote not responding
<i>enable</i>	<i>enabled – if not enabled, the value can be set, but the device value is not</i>
<i>Hrdwr_id</i>	<i>reference to SYS_AO table – the sys_ao tables do all interaction with the hardware</i>
<i>unittext</i>	<i>text for units(for display purposes)</i>
<i>value</i>	<i>in engineering units</i>
<i>zero</i>	<i>eng units; for calculating fracfs_value in sys_ao table</i>
<i>fullscale</i>	<i>eng units; for calculating fracfs_value in sys_ao table</i>
<i>Limitapp,limitref</i>	<i>Alarm handler of limit high/low</i>
<i>Hrdwrapp,hrdwrref</i>	<i>reference to detector, temperature controller, or pressure controller</i>

APPDO

persistent,

application_id integer references application(application_id),

id integer,

ezid integer,

name char,

io_status integer,

enable bool,

hrdwr_id char references sys_do(hrdwr_id),

value bool,

text0 char,

text1 char,

limitapp integer,

limitref integer ,

hrdwrapp integer,

hrdwrref integer,

auto_offtime integer,

timerref integer references dotimer,

foreign key(hrdwrapp,hrdwrref) references app_hardware(application_id,id),

primary key(application_id,id),

foreign key(limitapp,limitref) references limit(application_id,limit_id)

Usage: Entries are created in the Advance System Manager. The application uses a DO that is defined in one of the Sys_Do tables by selecting the Hrdwr_id. View this table on the **MMI[Menu][I/O][DO]** or the **SM[Application] [Application IO] [Appdo]**.

Name	Description
application_id	must be a valid application
id	reference in methods for this i/o – customer defined
<i>ezid</i>	<i>ezchrom id</i>
<i>name</i>	<i>customer defined</i>
Io_status	set by system 0 = normal -1 = not initialized -5= overscale error -2 = general error -6= not scanning -3= comm error -7= no hrdwr_id -4= open error -8= remote not responding
<i>enable</i>	<i>enabled – if not enabled, the value can be set, but the device value is not</i>
<i>Hrdwr_id</i>	<i>reference to sys_do table – the sys_do tables carry out all interaction with the hardware</i>
<i>value</i>	<i>on or off</i>
<i>text0</i>	<i>meaning of 0 value (for display purposes)</i>
<i>text1</i>	<i>meaning of 1 value (for display purposes)</i>
<i>Limitapplimitref</i>	<i>Alarm handler of limit high/low</i>
<i>Hrdwrapp,hrdwrref</i>	<i>reference to detector, temperature controller, or pressure controller</i>
<i>auto_offtime</i>	<i>seconds for turning off do automatically</i>
<i>timerref</i>	<i>reference to timer table</i>

APPAI

persistent,

application_id	integer,
id	integer,
name	char,
io_status	integer,
enable	bool,
hrdwr_id	char,
unittext	char,
value	float32,
zero	float32,
fullscale	float32,
limitapp	integer,
limitref	integer ,
hrdwrapp	integer,
hrdwrref	integer,

primary key(application_id,id),
foreign key(application_id) references application(application_id),
foreign key(hrdwrapp,hrdwrref) references app_hardware(application_id,id),
foreign key(hrdwr_id) references sys_ai(hrdwr_id),
foreign key(limitapp,limitref) references limit(application_id,limit_id)

Usage: Entries are created in the Advance System Manager. The application uses an AI that is defined in one of the **Sys_Ai** tables by selecting the Hrdwr_id. View this table on the MMI[Menu][I/O][AI] or the SM[Application][Application IO] [Appai]..

Name	Description
application_id	customer defined id
id	reference in methods for this i/o - customer defined
<i>name</i>	<i>customer defined</i>
Io_status	set by system 0 = normal -1 = not initialized -5= overscale error -2 = general error -6= not scanning -3= comm error -7= no hrdwr_id -4= open error -8= remote not responding
<i>enable</i>	<i>enabled? if disabled, the value can be set independent of the SYS_AI tables</i>
<i>Hrdwr_id</i>	<i>reference to sys_ai table</i>
<i>unittext</i>	<i>text for value display</i>
<i>value</i>	<i>in engineering units</i>
<i>zero</i>	<i>used to calculate to eng units from fracfs_value in sys_ai tables</i>
<i>fullscale</i>	<i>used to calculate to eng units from fracfs_value in sys_ai tables</i>
<i>Limitapp,limitref</i>	<i>Alarm handler of limit high/low</i>
<i>Hrdwrapp,hrdwrref</i>	<i>reference to detector, temperature controller, or pressure controller</i>

APPDI

persistent,

APPLICATION_ID	INTEGER REFERENCES APPLICATION(APPLICATION_ID),
ID	INTEGER,
name	char,
io_status	integer,
enable	bool,
hrdwr_id	char references sys_di(hrdwr_id),
value	bool,
text0	char,
text1	char,
limitapp	integer,
limitref	integer
hrdwrapp	integer,
hrdwrref	integer,

foreign key(hrdwrapp,hrdwrref) references app_hardware(application_id,id),

primary key(application_id,id),

foreign key(limitapp,limitref) references limit(application_id,limit_id)

Usage: Entries are created in the Advance System Manager. The application uses a DI that is defined in one of the Sys_Di tables by selecting the Hrdwr_id. View this table on the MMI[Menu][I/O][DI] or the SM[Application][Application IO] [Appdi].

Name	Description
application_id	customer defined id
id	reference in methods for this i/o - customer defined
<i>name</i>	<i>customer defined</i>
Io_status	set by system 0 = normal -1 = not initialized -5= overscale error -2 = general error -6= not scanning -3= comm error -7= no hrdwr_id -4= open error -8= remote not responding
<i>enable</i>	<i>enabled? if disabled, the value can be set independent of the SYS_AI tables</i>
<i>Hrdwr_id</i>	<i>reference to sys_di - the sys_di tables carry out all interaction with the actual hardware</i>
<i>value</i>	<i>on or off</i>
<i>text0</i>	<i>meaning of 0 value (for display)</i>
<i>text1</i>	<i>meaning of 1 value (for display)</i>
<i>Limitapp,limitref</i>	<i>Alarm handler of limit high/low</i>
<i>Hrdwrapp,hrdwrref</i>	<i>reference to detector, temperature controller, or pressure controller</i>

STREAM_METHOD

persistent,

application_id	integer,
stream_id	integer,
method_id	integer,
enable	bool,
automvr	bool,
autolog	bool,
status	char,
seqref	array of sequence_entry,
autotrt	bool,
lognow	bool,
mvrnow	integer,
trtnow	bool,
mvrpgm	integer,
primary key(application_id, stream_id, method_id),	
foreign key (application_id)references application(application_id)	

Usage: The database maintains this table of unique stream/method combinations for an application. It serves as a summary table for the sequence tables and gives the customer important support capabilities to disable certain streams/methods and automatically log and approve results. View this table on the MMI[Menu][Setup][Streams]] or the SM[Application] [Tables] [Stream_Method].

Name	Description
application_id	
stream_id	
method_id	
<i>enable</i>	<i>enabled? Disabling a stream/method is cause it to be skipped in the active sequence for a running application.</i>
<i>automvr</i>	<i>automatically approve(qualify) results if no alarm this cycle?</i>
<i>autolog</i>	<i>automatically log results at end of cycle if no alarm this cycle?</i>
Status	status IF in the active Sequence built from the screen status in the SEQUENCE_ENTRY table.
seqref	references to all sequence_entry rows
<i>autotrt</i>	<i>automatically transmit to hosts at end of cycle?</i>
<i>lognow</i>	<i>print log on demand</i>
<i>mvrnow</i>	<i>MVR on demand 1=normal 2=forced</i>
<i>trtnow</i>	<i>TRT on demand</i>
<i>mvrpgm</i>	<i>program that runs after automvr</i>

ALARMHANDLER

persistent,	
application_id	integer references applicatiion,
alarm_code	integer,
enable	bool,
programid	integer,
text	char,
dosetting	bool,
doapp	integer,
doref	integer,

primary key(application_id,alarm_code),
foreign key(doapp,doref) references appdo(application_id,id)

Usage: Application specific alarm handler. Entries here can override system alarm codes and specify program execution, printing and do setting. See Alarm Handling section for additional information.

Name	Description
application_id	
Alarm_code	900 – 996 are reserved for customer alarms
Enable	is alarm enabled? Can disable system alarms here.
programid	program to run when alarm occurs
Text	text will embed special symbols for parameter substitution(see Section IV. note ->first character is alarm type(!,?,+)followed by a blank %1 – application_id %2 – stream_id %3-%9 param3-param9
dosetting	value setting for do
Doapp,doref	do to set when alarm occurs

RESULT

persistent,

application_id	integer,
stream_id	integer,
result_index	integer,
result_type	integer,
status	integer,
method_id	integer,
channel	integer,
program_id	integer,
result_name	char,
value_units	char,
cycle_runtime	datetime,
buffered_value	float32,
saved_time	datetime,
saved_value	float32,
limitapp	integer,
limitref	integer,
aoapp	integer,
aoref	integer,
logval	integer,
trtval	integer,
host_euhi	float32,
decimal_places	integer,
doapp	integer,
doref	integer,

primary key(application_id,stream_id,result_index),
foreign key(limitapp,limitref) references limit(application_id,limit_id),
foreign key(doapp,doref) references appdo(application_id,id),
foreign key(aoapp,aoref) references appao(application_id,id),
foreign key(application_id,stream_id) references stream(application_id,stream_id)

Usage: Final results are moved from the EZChrom result tables or generated in a program. View this table on MMI[Menu][Recent Chrom/Results] or the SM[Application][Results]. Add/delete results in SM[Application][Tables][Results]

Name	Description
Application_id	
Stream_id	
Result_index	
Result_type	not used
Status	1= valid 2= Invalid
Method_id	method associated with result
Channel	channel associated with result
Program_id	Event associated with result
Result_name	name for result, if available
Value_units	text for value units
Cycle_runtime	end of data acquisition – matches chromatogram table
Buffered_value	preliminary value
Saved_time	timestamp for saved_value
Saved_value	approved value(qualified]
Limitapp,limitref	limit high/low for buffered_value
Aoapp,aoref	AO to set at end of cycle
logval	<i>result is marked for result logging – the value indicates the order</i>
trival	<i>result is marked for TRT – the value indicates the order</i>
host_euhi	<i>EUHI value for HCIH result transmission</i>
decimal_places	<i>decimal places to use for reporting; default 2.</i>
doapp,doref	DO to set at end of cycle

EXTRESULT

persistent,

application_id	integer,
stream_id	integer,
result_index	integer,
result_type	integer ,
status	integer,
result_name	char,
anlz_id	integer references analyzer,
remapp	integer,
remstr	integer,
resnum	integer,
Optimaname	char ,
limitapp	integer,
limitref	integer ,
adhref	integer shared references adhconnection,
polltimer	integer references iopolltimer(id),
saved_value	float32,
linkapp	integer,
linkstrm	integer,
linkref	integer,
resapp	integer,
resstrm	integer
resref	integer,

primary key(application_id,stream_id,result_index),
foreign key(application_id,stream_id) references stream(application_id,stream_id),
foreign key(linkapp,linkstrm,linkref) references extreslink(application_id,stream_id,result_index),
foreign key(resapp,resstrm,,resref) references result(application_id,stream_id,result_index)

Usage: Results that are required by an application's Visual Basic programs that come from another application on the analyzer, another stream in this application, an Advance Data Hiway unit, or an Advance Optima. . View and edit this table on SM[Application][Extresult].

Name	Description
<i>Application_id</i> <i>Stream_id</i>	
<i>Result_index</i>	
<i>Result_type</i>	<i>0 = adh, 1 = remote MaXum, 2 = local 3=Optima</i>
Status	-2 = error -1= not initialized 0 = ok
<i>Result_name</i>	<i>name for result, if available</i>
<i>Anlz_id</i>	<i>Analyzer to access remote MaXum or ADH</i>
<i>Remapp</i>	<i>Application on remote or local MaXum</i>
<i>Remstr</i>	<i>Remote Stream – for Maxum and ADH</i>
<i>Resnum</i>	<i>Remote result number – for Maxum and ADH</i>
<i>Optimaname</i>	<i>Name to extract from dvi_component_meas table of an Advance Optima unit</i>
<i>Limitapp.limitref</i>	<i>Alarm handler of limit high/low</i>
<i>Adhref</i>	for internal use
<i>Polltimer</i>	2 = slow 3 = medium 4 = fast
<i>Saved_value</i>	value
<i>Linkapp,linkstrm,linkapp</i>	for internal use only
<i>Resapp,ressstrm,resref</i>	Indicates result to update with saved_value

PARAMETER

persistent,
appcontext **integer,**
streamcontext **integer,**
PARAMETER_ID INTEGER,
parameter_name char,
value char,
qid integer,
qtype integer,
primary key(application_id, stream_id, parameter_id)

Usage: Certain system parameters are available at system startup. Customers may add system or application parameters. Parameters can be used for Visual Basic arguments. Viewed on the **MMI[Menu][Setup][Parameters]** or the **SM[Application] [Tables] [Parameter]**.

Name	Description
<i>Appcontext</i>	<i>An application here means that this parameter is specific to this application, -1 is reserved for all applications</i>
<i>streamcontext</i>	<i>Stream id from application .-1 is reserved to indicate all streams.</i>
<i>Parameter_id</i>	<i>Unique id – customer assigned</i>
<i>Parameter_name</i>	<i>Customer defined name for parameter</i>
<i>Value</i>	<i>Customer defined value used as a constant</i>
<i>qid</i>	<i>id in queried table</i>
<i>qtype</i>	<i>table for query</i> <i>null = use parameter value as a constant</i> <i>0=none 3= buffered result</i> <i>1= AI 4= saved result</i> <i>2= DI 5= external result</i>

Class Exercises

1. Make a new program **hello.bas**. Your program should print “hello” using the **msgbox** statement. Connect to a Syscon. Run on the workstation. Make an .exe file. Attach program to database. Run on the Syscon.
2. Make a new program **results.bas**. Using a snapshot, select all the records in the result table and print the **result_name** and **saved_value**.
3. Make a new program **appdo.bas** . Use the table object, edit, update to set **appdo** 1000 enabled and set to true. Do the same with the **Execute SQL** method.
4. Make a new program to transmit results to a host.
5. Make a program to normalize results.
6. If **appdi** 1001 is false, get the **cycle_runtime** from the first result, store the hour and minutes in a result 9. Find the adjustment hours in the parameter table(1200) and adjust this time.

Exercise 1

```
Sub main()  
  MsgBox "hello"  
End Sub
```

Exercise 2

```
Sub main()  
  Dim mysnap As Snapshot  
  Dim sqlstr As String  
  sqlstr = "select application_id,stream_id,result_index,result_name,saved_value"  
  sqlstr = sqlstr + " from result order by application_id,stream_id,result_index"  
  Set mysnap = db.CreateSnapshot(sqlstr)  
  mysnap.MoveFirst  
  Do While Not mysnap.EOF  
    MsgBox mysnap(3) + " " + Str$(mysnap(4))  
    mysnap.MoveNext  
  Loop  
End Sub
```

Exercise 3A

```
Sub main()  
  Dim mytab As Table  
  Dim sqlstr As String  
  Dim tr As String  
  tr = "TRUE"  
  sqlstr = "select application_id,id,enable,value"  
  sqlstr = sqlstr + " from appdo where application_id=" + Trim$(Str$(App1))  
  sqlstr = sqlstr + " order by id"  
  Set mytab = db.OpenTable(sqlstr)  
  mytab.MoveFirst  
  Do While Not mytab.EOF  
    If mytab(1) = 1000 Then  
      mytab.Edit  
      mytab(2) = tr  
      mytab(3) = tr  
      mytab.Update  
    End If  
    mytab.MoveNext  
  Loop  
  mytab.Close  
End Sub
```

Exercise 3B

```
Sub main()  
Dim sqlstr As String  
  Dim tr As String  
  tr = "TRUE"  
  sqlstr = "select application_id,id,enable,value"  
  sqlstr = sqlstr + " from appdo where application_id=" + Trim$(Str$(AppI))  
  sqlstr = sqlstr + " order by id"  
  Set mytab = db.OpenTable(sqlstr)  
  mytab.FindFirst("id=1000")  
  If Not mytab.NoMatch Then  
    mytab.Edit  
    mytab(2) = tr  
    mytab(3) = tr  
    mytab.Update  
  End If  
  mytab.Close
```

End Sub

Exercise 3C

```
Sub main()  
  
  Dim n As Long  
  Dim sqlstr As String  
  sqlstr = "update appdo set enable=true,value=true where application_id = " + Trim$(Str$(AppI))  
  sqlstr = sqlstr + " and id=1000"  
  n = db.ExecuteSQL(sqlstr)
```

End Sub

Exercise 4A

```
Sub main()  
'Program to TRTNOW for a stream  
' Functionality : This program finds the first stream_method entry for a stream and does  
' a TRTNOW - transmit results to host  
' Configuration : No dependencies On parameter Table  
' Only assumption is stream  
Dim sqlstr As String  
Dim StrTrue As String  
Dim TblStrmMeth As Table  
Set StrTrue = "TRUE"  
' create the SQL string for stream_method table  
sqlstr = "SELECT application_id, stream_id, method_id, trtnow"  
sqlstr = sqlstr + " FROM stream_method"  
sqlstr = sqlstr + " WHERE stream_id = " + Trim$(Str$(Strm))  
'get table  
Set TblStrmMeth = db.OpenTable(sqlstr)  
TblStrmMeth.MoveFirst  
If Not TblStrmMeth.EOF Then  
  TblStrmMeth.Edit  
  TblStrmMeth(3) = StrTrue  
  TblStrmMeth.Update  
End If  
  TblStrmMeth.Close  
End Sub
```

Exercise 4B

Sub main()

Dim n **As** Long

Dim sqlstr **As** String

sqlstr = "update stream_method set trtnow = true where stream_id" + **Trim\$(Str\$(Strm))**

n = db.**ExecuteSQL**(sqlstr)

End Sub

Exercise 5

Sub main()

'Program to Normalize results

' Only assumption is current stream

Dim sqlstr **As** String

Dim TimeStamp **As** String

Dim Rsum **As** Double

Dim TblResult **As** Table

Dim n **As** Integer

' check if cal stream

If Strm >= 30 **Then** 'Exit if Cal stream

Exit Sub

Endif

' create the SQL string for result table

sqlstr = "SELECT application_id,stream_id,result_index,buffered_value,cycle_runtime"

sqlstr = sqlstr + " FROM result"

sqlstr = sqlstr + " WHERE"

sqlstr = sqlstr + " application_id = " + **Trim\$(Str\$(Appl))**

sqlstr = sqlstr + " AND stream_id = " + **Trim\$(Str\$(Strm))**

'get table

Set TblResult = db.**OpenTable**(sqlstr)

Rsum = 0.0

' sum results

TblResult.**MoveFirst**

Do Until TblResult.**EOF**

Rsum = Rsum +TblResult(3)

TblResult.**MoveNext**

Loop

MsgBox "sum = "+**Str\$(Rsum)**

TblResult.**MoveFirst**

'normalize results

Do Until TblResult.**EOF**

TblResult.**Edit**

TblResult(3) = (TblResult(3) / Rsum * 100)+.00001

MsgBox Str\$(TblResult(3))

TblResult.**Update**

TblResult.**MoveNext**

Loop

TblResult.**Close**

End Sub

Exercise 6

Sub main()

'Functionality : this program records the time whenever a grab sample is taken by

' setting a timestamp for a result based on the state of a DI.

' Assumptions: the timestamp of the first result is used

' DI 1001 is examined,if false, buffered_value and saved_value

' of result 9 is set to day.hrmin

' 1501 means 15:01

Dim Dltbl **As** Table

Dim Resulttbl **As** Table

Dim ParaTbl **As** Table

Dim TimeStamp **As** String

Dim Sql **As** String

Dim SetToTrue **As** String

Dim SetToFalse **As** String

Dim resultvalue **As** Variant

Dim resdbl **As** Double

Dim Hr **As** String

Dim corfactor **As** Double

' boolean values as strings

SetToTrue = "TRUE"

SetToFalse = "FALSE"

' select parameter that contains time correction factor

sql = "SELECT appcontext,streamcontext,parameter_id,value"

sql = sql + " FROM parameter WHERE appcontext = " + Trim(Str\$(Appl))

sql = sql + " AND streamcontext = -1 AND parameter_id = 1200"

Set ParaTbl = db.**OpenTable**(sql)

ParaTbl.**MoveFirst**

If ParaTbl.**EOF** **Then**

MsgBox "no time correction parameter"

Exit Sub

Endif

corfactor = ParaTbl("value")

' **MsgBox** "corfactor = " + Str\$(corfactor)

' Create the SQL string for DI table

sql = "SELECT application_id,id,value,enable"

sql = sql + " FROM appdi"

sql = sql + " WHERE"

sql = sql + " application_id = " + Trim(Str\$(Appl))

sql = sql + " and id = 1001"

' get the DI table

Set Dltbl = db.**OpenTable**(sql)

' Create the SQL string for result table for getting timestamp

' get the time stamp from any result, say result_index = 1

sql = "SELECT application_id,stream_id,result_index,buffered_value,saved_value,cycle_runtime"

sql = sql + " FROM result"

sql = sql + " WHERE"

sql = sql + " application_id = " + Trim(Str\$(Appl))

sql = sql + " AND stream_id = 1"

' get the result table, we will get only one record

Set Resulttbl = db.**OpenTable**(sql)

Resulttbl.**MoveFirst**

TimeStamp = Resulttbl(5) ' save the time stamp

' **MsgBox** timestamp

hr = **Mid**\$(timestamp,13,2)+**Mid**\$(timestamp,16,2)

resultvalue = **Cdbl**(hr) - corfactor

```
resdbl = resultvalue
' MsgBox Str$(resdbl)
Resulttbl.MoveFirst
Resulttbl.FindFirst("result_index = 9")
Dltbl.MoveFirst
if Dltbl.EOF Then
  MsgBox "DI 1001 not available"
  Exit Sub
Endif
if Dltbl(2) = SetToFalse Then
  Resulttbl.Edit
  Resulttbl(3) = Resdbl+ .00001 ' set the result
  Resulttbl(4) = Resdbl+ .00001 ' set the result
  Resulttbl(5) = TimeStamp 'set the time stamp too
  Resulttbl.Update
Endif
Resulttbl.Close ' close the result table
Dltbl.Close ' close the DI table
End Sub
```