SQAPL Version 5.0 Release Notes

Document version dated 2005-06-09, companion of DLL version 5.0.2.12

This document describes functionality added to SQAPL / APL Link since the last major release, which was numbered version 3.0.14 (December 1999). A number of updates have been released between these two versions, but 5.0 is the first version to have significant new functionality.

Overview

This section provides an overview of new or changed functionality of SQAPL version 5.0. A few of the changes have a small likelihood of requiring changes to existing applications, these changes are discussed first. We believe that very few applications will require change: No beta test customers have reported problems. However, you should carefully read the discussion before using SQAPL version 5 in production applications.

More details on each feature are provided in the section following the overview.

Note: All examples assume `⎕io←1`.

ODBC Version 3.51

Version 5.0 has been upgraded to the most recent ODBC API, version 3.51, in order to give users access to the most up-to-date features of the newest ODBC drivers.

Compatibility issue: Three ODBC data type numbers have changed, and we decided to follow ODBC and use the new numbers rather than stick to the old ones. We believe that a small number of SQL code generation applications might be using ODBC data type numbers. These applications will need slight modifications in order to use version 5 (in a nutshell, replace the numbers 9, 10 and 11 by 91, 92 and 93).

Exception Handling

In previous versions, exceptions occurring within the SQAPL DLL or an ODBC driver would cause APL to crash, with little or no information about the cause. In version 5, exceptions are trapped and signalled as APL errors by the function `⍙SQAPL`. In the event of a failure in or below the level of the DLL, you will see something like the following:

```
[SQAPL] Exception c0000005 at 642574b Memory read at 10
ExecDirect[3] r←⍙SQAPL'ExecDirect'y
                ^
```

The APL Event Number signalled is 11 (same as a DOMAIN ERROR). The bug in question, found during testing, has been fixed, there are currently no known examples which will cause this type of error.

Configuration Free Installation

For most installations, version 5 can make do without character set files and configuration entries in the registry. This should simplify the creation of installation scripts for applications which use SQAPL.

If you need a translate table different from the default provided by SQAPL, a new function `SetXLate` allows you to set the translate table you want to use under programme control, rather than using external registry entries and configuration files.

Compatibility issue: A new warning number 10091 is issued when the `SQAInit` function decides to use default translate tables because it is unable to find configuration files. You may need to decide whether or not to ignore this error message.

Retrieval of Multiple Warning Messages

Previously, if a single operation caused multiple warnings, `SQAGetWarning` only retrieved the "most important" one (the first one). In version 5, `SQAGetWarning` returns all warnings issued by ODBC.

Compatibility issue: In the event that more than one message is issued, the second element of the result of `SQAGetWarning` will contain more than one element. It was already a vector in previous versions, but always had length 1.

Enhancements to SQAX

The `SQAX` function allows repeated execution of the same prepared statement on each row of a bind value matrix. Most errors occurring during execution are reported as warnings rather than errors, because the call to `SQAX` may have been successful even if certain input values caused errors. These error messages are retrieved using `SQAGetWarning`. In version 5, each error message includes a new 4th element, which is the row number for which the message was issued.

Version 5 also adds a new option called `StopOnError`, which allows you to request that `SQAX` not continue processing, but stop and report first problem encountered as an error rather than a warning.

Version 5 has also been enhanced so that does a better job of detecting whether an ODBC driver really supports so-called "parameter sets". Previously, the use of `SQAX` with MS Access or MySQL drivers required use of the `('Loop' 0)` on `SQAPrepare` to instruct SQAPL that the driver was unable to loop on the data itself.

Compatibility issue: See the next section titled *Change to All Error Results*.

Change to All Error Results

In the event of an error, all functions now return a result with 4 elements instead of the 3 which were returned in earlier versions. The new fourth element contains an index to the operation which failed, and is currently only different from zero when using `SQAGetWarning` following `SQAX`, where it indicates the row in the bind value matrix for which the error or warning was issued.

Compatibility issue: In the unlikely event that you have code which depends on there being exactly three elements in the result following an error, you will need to make a change.

Change to Result of SQADescribe

Compatibility issue: The length of `(2 1 8⊃0 SQADescribe)` on a Connection, a bit vector which documents ODBC function calls supported by the driver, now has around 4000 elements, previously it had under 100 (the exact length is not documented here because you should not write code which relies on this – it may change again in the future!). This is because Microsoft started adding offsets of 1000 to function numbers supported by newer ODBC versions.

SQAExecDirect

This new function makes it possible to prepare and execute a statement in a single call.

Faster Fetches

A new “Columnwise” option returns data a column at a time, which allows SQAPL to allocate memory once for the entire result, rather than allocating and formatting an APL array for each cell in the result. In some cases, speedups of 10x have been observed.

New Date Options

The numeric date representation used by SQAPL is “the number of days since 1900-01-01”. The first of January 1900 is (obviously, therefore) day number zero. Other programming tools tend to give this day the number 1, and the most common standard (used by Excel) also contains bug, giving the non-existent day 1900-02-29 the number 60. Someone forgot that 1900 was not a leap year! Version 5.0 now supports “all three” numbering systems.

Variable Descriptions Separate from SQL Statement

Previously, all input and output variable descriptions had to be embedded within the SQL statement being prepared. In version 5, you can decide to provide this information using a matrix separate from the SQL, and using an SQL statement more similar to that used with other programming tools.

Ability to Open a Data Source in Read Only Mode

A new option for `SQAConnect` allows you to make sure that you do not inadvertently modify data. This makes it safer to allow users to enter their own SQL queries in your application. With some drivers, this option probably also improves performance – this has not been verified.

SQLNativeSQL

We have added support for an ODBC call which allows you to see the native SQL statement which your ODBC SQL statement was translated into.

Translation of Binary Buffers

In version 5, you can specify that binary data be translated between the APL and OS character sets.

More Tolerant SQL Parser

Version 5 allows you to use multi-line SQL statements (statements including “New Line” characters). Also, the sequence `::` (two colons) is now allowed to pass through, instead if causing an error message because it was interpreted as an erroneous variable declaration. These two changes intended to make it easier to use Stored Procedures.

SQA Namespace

Under Dyalog APL, the `SQAPL` workspace now contains a namespace `SQA`, which contains functions with names without the leading `SQA`. A set of root level functions is also provided, these functions all call into the `SQA` namespace. For example, the function `#.SQAConnect` calls `SQA.Connect`.

SQATEST Desupported

The bad news: The test suite which puts the system through its paces is not distributed with version 5.0. It may be reintroduced or made available for download at a later date. Contact Insight Systems if you think you need a test tool and we’ll send you the latest internal version.

Details

ODBC Version 3.51

SQAPL version 5.0 uses version 3.51 of the Microsoft's Open Database Connectivity interface (ODBC). This means that you will have access to the latest functionality of any ODBC drivers which you are using. In general, we believe that this will improve performance, but of course, access to new versions of drivers may also bring unexpected changes in functionality along with it. You will need to re-test your application, but except for the data type name changes described below, no changes are *expected*.

If your application is calling (`0 SQADescribe` …) on prepared cursors, and referencing the ODBC data type number (the 7th element of the description of each input or output variable), you may need to take the changed data type numbers into account. In this case, you can simply refer to the 5th element, called "subtype", which was previously identical to the 7th element in earlier versions, but is now alone in containing the old type numbers. If you have been calling `SQATypeInfo` on connections, and using the `DATA_TYPE` column for any purpose, for example to pick a data type when creating a table, you should expect these data type numbers to change.

In all cases, you should be able to get the same behaviour as before by changing references to data type numbers 9 (Date), 10 (Time) and 11 (Timestamp) to 91, 92 and 93, respectively.

More information is available from Microsoft online, in the ODBC Programmers Reference, in particular the page:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odbc/htm/odbcdatetime_data_type_changes.asp

The small behavioural differences between the data types are handled by SQAPL, so the only changes required to APL applications should be where code explicitly uses the data type numbers to control application behaviour.

Configuration Free Installation

By far the most common problem encountered by users of SQAPL has been a change in the registry or location of the software, resulting in the inability of the system to locate its configuration files. Because SQAPL translates function names, and all error messages have to be translated between the APL system "Atomic Vector" and the DLL environment (which is most often using "ANSI"), a failure to find the tables resulted in cryptic numeric error messages like (20 10003), with no accompanying text.

For most users, version 5 will work without the configuration files, and it will always be able to produce textual error messages in the event of a problem. If your application has been using the standard `APLUNICD.INI` file as shipped, and if your application has not been using the binary APL data format to exchange APL objects with other APL systems, you probably do not need to know any more about this change to the system, except that you can safely ignore the following warning from `SQAInit` if you ever see it:

```
      SQAInit ''
¯1  10091  Using default translate table: Cannot read APL_UNICODE
  from Environment
```

If you have been using a non-standard translate table, need to exchange APL objects between different APL systems, or cannot help yourself and are curious about how all this works, please read **Appendix A: More about Character Translation**, for more details.

If, after reading the appendix, you decide that you would prefer to establish translate tables under the control of your application, rather than using the configuration files or the defaults which are established if no files are found, you can use the function `SetXLate` to achieve do this in the event that `SQAInit` return the `¯10091` error:

```
      SetXLate APLName HostName AV2HOST HOST2AV
0
```

The first parameter is a 6-element name of your translate table – this is only used to identify your character set in the event that you write APL objects as SCARs (in bind variable format “Z”). To be useful, the name must identify a translate table in the aplunicd.ini file used by a system which attempts to read your array (unless the reading system is using the same name, in which case translation is unnecessary).

The second parameter – the Host Name – is not currently used.

The two last parameters are 256-element vectors which define the translate tables used to translate data from APL to the host, and vice versa.

The current translation settings are returned by `0 SQADescribe` on the root object:

```
      ⍴¨(2 1⊃0 SQADescribe '.')[6 7 8 9]
6 6 256 256
```

To see an example of the use of `SetXLate`, look at the last few lines of `SQAInit`, which uses the function to establish a default translate table in the event that configuration files are not found.

Retrieval of Multiple Diagnostic Messages

In version 5.0, `SQAGetWarning` returns all warnings generated by the last operation. Previously, if a single operation caused multiple errors, `SQAGetWarning` only retrieved the “most important” one (the first one). For example, in version 3.x, a connection to SQL Server might give the following results:

```
      SQABrowseConnect 'C1'
¯1
      ⎕←z←SQAGetWarning 'C1'
0  4  01000  5701  [Microsoft][ODBC SQL Server Driver][SQL Serv
er]Changed database context to 'master'.
      (⍴2⊃z) (⍴¨2⊃z)
1  3
```

With version 5:

```
      z←SQAGetWarning 'C1'
      (⍴2⊃z) (⍴¨2⊃z)
2  3 3
      ↑2⊃z
4  01000  5701  [Microsoft][ODBC SQL Server Driver][SQL Server]
Changed database context to 'master'.    0
4  01000  5703  [Microsoft][ODBC SQL Server Driver][SQL Server]
Changed language setting to us_english.  0
```

If you have code which assumes that the second element of the result will contain a single element, you may need to modify the code slightly to pick the first element of the new result. The old result should be the first element in the new list.

Enhancements to SQAX

In version 5, each message returned by `SQAGetWarning` following `SQAX` includes a new 4th element, which is the row number for which the message was issued.

```
      SQAPrepare 'C1.I1' 'Insert into v5demo(coldt) values(:<S:)'
                         ('Bulk' 3)
0
      SQAX 'C1.I1' (3 1⍴7↑¨(2005 2 28)(2005 2 29)(2005 3 1))
¯1 2 3
```

The first element of the result of `SQAX` is `¯1`, indicating that warnings were issued. The second element tells us that a total of 2 rows were modified, and the final element that the last row of input used was the 3rd (the final element is obsolescent; it will always be equal to the number of input rows).

```
       ⍴¨z←2⊃SQAGetWarning 'C1.I1'
4
       (⍳4),[1.5]1⊃z

1                                                    4
2                                              22007  35
3  [Microsoft][ODBC Microsoft Access-driver]Invalid Date
4                                                    2
```

A new option called `StopOnError` can be used to instruct `SQAX` to stop on the first problem and report it as an error rather than a warning:

```
      SQAPrepare 'C1.I1' 'Insert into v5demo(coldt) values(:<S:)'
                       ('Bulk' 3)('StopOnError' 1)
0
      SQAX 'C1.I1' (3 1⍴7↑¨(2005 2 28)(2005 2 29)(2005 3 1))
4   22007  35  [Microsoft][ODBC Microsoft Access-driver]Invalid D
ate  2
```

Note that the 4th element in the result (2 in above example example) indicates the row number which failed. The above result is identical to the first element of (the second element of) the result of `SQAGetWarning` in the previous example. Although an error is returned, the first set of input values was used, so the table has been modified (depending on transaction options, naturally).

The bad news: Unfortunately, testing during the development of version 5 has revealed that ODBC drivers do not behave consistently with respect to this feature. Drivers disagree about whether to use 0 or 1 origin when reporting the row numbers. Also, some drivers simply ignore the request to stop on the first error. You should experiment with the ODBC drivers which you are using (but beware that new releases of the drivers may have altered behaviour). You can achieve more reliable behaviour by specifying `('Loop' 0)` in the argument to Prepare, which instructs SQAPL not to use the ability of drivers to loop on multiple sets of input data, even if they report that they support it. However, this may have a performance impact. With `('Loop' 0)`, indices will always be origin 1 and `SQAX` will always stop on the first error.

Change to All Error Results

In the event of an error, all functions now return a result with 4 elements instead of the 3 which were used in earlier versions. The new fourth element contains an index to the operation which failed, and is currently only different from zero when using `SQAX` (or `SQAGetWarning` following `SQAX`), where it identifies the row in the bind value matrix for which the error or warning was issued.

Compatibility issue: In the unlikely event that you have code which depends on there being exactly three elements in the result following an error, you will need to make a change. In the event that beta-testers report numerous problems due to this change, we will possibly withdraw this enhancement, or limit it to the results of `SQAX` and `SQAGetWarning`. At this point we have elected to use 4 elements everywhere in the name of consistency.

Exception Handling

In the event of a failure in or below the level of the SQAPL.DLL, an APL error will be signalled as follows:

```
[SQAPL] Exception c0000005 at 642574b Memory read at 10
ExecDirect[3] r←⍙SQAPL'ExecDirect'y
                 ^
```

The event number signalled is 11 (DOMAIN ERROR). However, we do not recommend that you attempt to automate recovery by using error trapping, because the internal state of SQAPL and any ODBC drivers you have been using is unknown following an exception. If you must restart, you should at least unload and reload the DLL using `SQAClose '.'` followed by a new call to `SQAInit`.

SQAExecDirect

You can prepare and execute a statement in a single call using this new function, which is primarily aimed at calling stored procedures:

```
      SQADo 'C1' 'create table tenkrows(C1 CHAR(10), C2 INTEGER)'
0  C1.s2
      data←(↓'ZI10' ⎕FMT ,[1.5]⍳10000),[1.5]⍳10000
      stmt←'insert into tenkrows values (:X<C10:,:Y<I:)'
      SQAExecDirect 'C1.i1' stmt data ('Bulk' 10000)
0 10000 10000
      SQAClose 'C1.i1'
0
```

Note that, since the function combines `SQAPrepare` and `SQAExec`, all options for these two functions are valid as options of `SQAExecDirect`. Also note that, following the call, the cursor is still open - even if an error was returned. You should call `SQAFetch` if any data was returned, and finally `SQAClose`.

Faster Fetches

If the new `Columnwise` option is set to `1`, `SQAFetch` returns data with each column as a simple matrix, saving the overhead of allocating an APL result element for each cell:

```
      stmt←'select * from tenkrows' ⍝ Data from previous example
      z←SQAExecDirect 'C1.I1' stmt
      ⍴2 1⊃z←SQAFetch 'C1.I1'
10000 2
```

But:

```
      z←SQAExecDirect 'C1.I1' stmt
      ⍴2⊃z←SQAFetch 'C1.I1' ('Columnwise' 1)
2
      ⍴¨2⊃z
 10000 10  10000
```

The first element is a 10-column character matrix, the second an integer vector. Numeric columns are returned as vectors, all other columns including multi-element date and time representations are returned as matrices.

Speedups of up to 10 times have been observed, your mileage will vary depending on how quickly your server and network can deliver the data, the number of columns extracted, and so forth.

Null flags are all returned as vectors in the 3rd element of the result, if the `Nulls` option is selected. Where a value was Null, SQAPL will set numeric values to 0 and character values to a row of spaces.

Note: `SQAConnect` also supports the `Columnwise` option, allowing you to set a default for the connection. For example:

```
      SQAConnect 'C1' 'MySQL' 'password' 'userid'
                ('MaxRows' 1000) ('Columnwise' 1)
0
```

If the option is set on the connection, it will control the shape of the data returned by `SQADo`.

New Date Options

The numeric date representation used by SQAPL is "the number of days since 1900-01-01". The meaning of this description is (as any APL programmer should agree) that first of January 1900 is day number zero. However, in order to make it easier for you to adapt to the various incorrect date numbering systems adopted by users of other programming languages, the `J` data type now takes a modifier which follows a hash (#) sign. The number following the modifier is the offset to the "zero base" date numbering system, except that for `J#2`, the difference is only 1 for dates in January and February 1900 (this is the numbering system used by Excel, also known as the "OLE Date Time").

```
     SQADo 'C1' 'create table dtdemo (ch1 char(10),dt1 datetime)'
0  C1.s2
     ⎕←data←3 2⍴'Jan 1' '1900-01-01' ...
 Jan 1   1900-01-01
 Feb 28  1900-02-28
 Mar 1   1900-03-01
      SQAExecDirect 'C1.I1' 'insert into dtdemo
                              values(:X:,:Y:)' data ('Bulk' 3)
0 3 3
      3⊃SQADo 'C1' 'select ch1,dt1,dt1,dt1,dt1,dt1
                        from datedemo :>C10,S,J,J#0,J#1,J#2:'
 Jan 1       1900 1 1 0 0 0 0    0  0  1  1
 Feb 28      1900 2 28 0 0 0 0  58 58 59 59
 Mar 1       1900 3 1 0 0 0 0   59 59 60 61
```

Variable Descriptions Separate from SQL Statement

In version 5, you can provide an 8-column matrix as the 3rd element of the argument to *SQAPrepare* or the 4th element of the argument to *SQAExecDirect*, (alternatively as an option named *BindInfo*) and declare the types of variables in this way, instead of using the "In Line" method for declaring variables.:

```
[;1] Direction (1=Bind, 0=Select)
[;2] Index (into Bind or Select?)
[;3] APL Type Number (1=C,2=I,3=F,X,O,Z,D,T,S,Y,H,12=J)
[;4] Buffer Type Number
[;5] Precision
[;6] Scale
[;7] Partial flag
[;8] Variant
```

A slight variation on the above example, with the addition of a bind variable, could have been written as follows:

```
      info←7 8↑(7↑1),(1,⍳6),[1.5]12 1 9 12 12 12 12
      info[6 7;8]←1 2 ⍝ J# variant
      info[2;5]←10    ⍝ C10
      info
1 1 12 0  0 0 0 0 ⍝ J bind variable (input)
0 1  1 0 10 0 0 0 ⍝ C10
0 2  9 0  0 0 0 0 ⍝ S
0 3 12 0  0 0 0 0 ⍝ J
0 4 12 0  0 0 0 0 ⍝ J#0
0 5 12 0  0 0 0 1 ⍝ J#1
0 6 12 0  0 0 0 2 ⍝ J#2
      SQAPrepare 'C1.S2' 'select ch1,dt1,dt1,dt1,dt1,dt1
                               from datedemo where dt1=?' info
0
      SQAExec 'C1.S2' 59
0 0
      SQAFetch 'C1.S2'
0    Mar 1       1900 3 1 0 0 0 0  59 59 60 61       6
```

The ODBC SQL syntax for bind variables is a question mark (?). Some databases, including Oracle, also allow the use of a colon followed by a name.

A new utility function *SQAParse* takes a statement with embedded variable declarations as its argument and returns a statement containing ? markers and the *BindInfo* matrix:

```
      'expr' 'info',[1.5] SQAParse 'select * from table
                                 where key1=:<I: and key2=:<C10:'
 expr  select * from table where key1=? and key2=?
 info  1 1 2 0  0 0 0 0
       1 2 1 0 10 0 0 0
```

Ability to Open a Data Source in Read Only Mode

A new option for *SQAConnect* allows you to make sure that you do not inadvertently modify data. With some drivers, this option probably also improves performance – this has not been verified.

```
      SQAConnect 'C2' 'SQATEST' ('ReadOnly' 1)
0
      SQADo 'C2' 'insert into sqatst(I1) values (1)'
```

```
4   HY000  ¯3035  [Microsoft][ODBC Microsoft Access-driver] Handl
ingen skal bruge en opdaterbar forespørgsel.
```

The error message states (in Danish) that the action requires an updateable query. In read only mode, such queries are hard to find.

SQANativeSQL

SQANativeSQL is a new function which can be applied to a prepared statement now returns the “native” SQL statement which the ODBC Driver has generated based on your ODBC SQL statement:

```
      SQAPrepare 'C1.S1' 'select * from datedemo'
0
      SQANativeSQL 'C1.S1'
SELECT *

FROM datedemo;
```

The result is a character vector. The above example used Microsoft Access; the “Access SQL” statement contains embedded carriage returns. You are in dire straits if this is really useful to you as a diagnostic tool, hopefully you will only ever use it to satisfy your curiosity.

Translation of Binary Buffers

Earlier versions always assumed that data which you were transfering to or from a binary buffer was binary data which should therefore not be translated between the Atomic Vector of the APL system and the external buffer. However, there are circumstances where you may want to perform this translation. In version 5, you can specify that your binary data also be translated, by using the #1 variant of the X data type.

Note: This is functionality has no effect under APL+, because no translation is required, the following example only makes sense under Dyalog APL:

```
      SQADo 'C1' 'create table bindemo (col1 binary)'
0  C1.s1
      SQADo 'C1' 'insert into bindemo values(:x<X256#1:)' ⎕av
0  C1.s1
      SQADo 'C1' 'insert into bindemo values(:x<X256:)' ⎕av
0  C1.s1

      out in←(2 1⊃0 SQADescribe '.')[8 9] ⍝ Translate tables

      out ⎕AV≡,3 1⊃SQADo'C1' 'select * from bindemo :>X256:'
1
      ⎕AV in≡,3 1⊃SQADo'C1' 'select * from bindemo :>X256#1:'
1
```

The first row inserted contains the atomic vector translated using the APL to Host translate table, because we specified X256#1. The second row contains the untranslated bytes from 0 to 255. In the first call to *SQADo* we read this information without translation and get the output translate table and ⎕*AV*, in the second call we request translation on reading and get ⎕*AV* and the input translate table.

More Tolerant SQL Parser

Version 5 allows you to use multi-line SQL statements (statements including “New Line” characters). Also, the sequence **::** (two colons) is now allowed to pass through, instead of causing an error message because it was interpreted as an erroneous variable declaration. These two changes intended to make it easier to use Stored Procedures. For example, if you define a stored procedure with embedded CR,LF sequences, some SQL database systems will refer to line numbers when errors are found.

Appendix A: More about Character Translation

SQAPL performs two different types of translation of character data:

- **All character data** must be translated between the APL Atomic Vector and the operating system environment (ANSI).
- **APL objects** containing text may need to be translated from the APL Atomic Vector of the APL system which has saved a binary APL object in an ODBC table or created an array used as the argument to *SQAScarToApl*, to the character set used by the receiving system.

The first form of translation is always required: Whenever an item of textual information is moved between the APL system and a database or back, character data must be translated. This requires knowledge of character set used by the APL system and that used by the host.

The second form of translation is only used when you read an APL object originating in a different APL system and containing character data, from the SCAR (Self Contained ARay) format to an APL variable in the current workspace. This can happen when you use an output variable in "Z" format, or when you call the function *SQAScarToApl*. This translation requires knowledge of the character sets used by the sending and receiving APL systems.

The file APLUNICD.INI contains definitions of a all the different characters sets known to SQAPL: At least one for each APL platform to which SQAPL has been ported (with some national variations), and for completeness, one representing the operating system, named ASCII, defined as the first 256 UNICODE characters, also known as ANSI. Each character set is defined as a list of 256 UNICODE characters. Using these tables, SQAPL is able to translate text between any two character sets.

The standard distribution version of this file starts with the section

```
[Charsets]
APL2=IBMA
APLIII=MAN3
APLUNX=MAN3
APLWIN=MAN3
DYALOG=DYA_IN
SAXAPL=SAX_US
HOST=ASCII
```

The names on the left identify a particular implementation (or "port") of SQAPL, plus one entry for the host operating system. On the right is the name of the table defining the alphabet which is used by the named environment.

The body of the APLUNICD file defines the individual tables. If you know what you are doing, you can create and use your own tables. This requires care; if you are creating a new table, we strongly urge you to contact Insight Systems to agree on the name and content of new tables.

When a SCAR object is created, character data is stored without translation (it is stored as a sequence of bytes which are indices into the atomic vector). Part of the header of the SCAR object names the translate table which was in use by the system which created the SCAR. The system which reads the SCAR checks the header, and if translation is necessary SQAPL needs to be able to locate BOTH tables (that of the reader as well as the writer) in the APLUNICD.INI file, in order to create the translate table between the two systems and perform the translation.

Locating the Required Tables

SQAPL searches the registry to find the string `APL_UNICODE`, which is used to locate the file `aplunicd.ini` and `SQAPLPATH`, which identifies the path in which `sqapl.ini` is located. The right argument to *`SQAInit`* identifies the registry key which should be searched, if no key is provided, the default is HKEY_CURRENT_USER\Software\Insight\SQAPL.

In versions of SQAPL prior to 5.0, any attempt to use SQAPL would fail with a numeric error code 20 10004 and no accompanying error message (because it could not be translated to APL), if the registry did not correctly identify the required files.

In version 5.0, SQAPL will search the path from which the DLL was loaded, if it cannot find these entries in the registry, so the registry entries are no longer required if the files are stored in the same path as the DLL.

If the files are not found, *`SQAInit`* will establish a default translate table between APL and the DLL using other techniques (*`⎕NXLATE`* in Dyalog APL, no translation is actually required for APL+Win because it contains all non-APL characters in the correct ANSI positions). A warning will be returned, for example:

```
      SQAInit ''
¯10091  Using default translate table:  Can not read APL_UNICODE
from Environment
```

If your application was using the default translate table and had no special national requirements, and you do not intend to transfer APL objects in SCAR format, you can ignore this error message, translation of textual data between APL and SQL databases will be done correctly.

See the "Details" section for information about how to establish your own translation table using the function *`SetXLate`*, if you would like to do this without using aplunicd.ini.

Appendix B: Change Log between 3.0.14 and 4.0.1

Version Date	Change Description
4.0.1 & 3.0.28 2004-03-23	Added handler for all exceptions: returns an error to APL. MaxRows set on a cursor were overwritten by Prepare Report an error rather than crash on statements which were not a char array.
3.0.24 2004-01-28	SQAExec sometimes failed on ‘Bulk’ statements. SQAExec ‘c1.s1.b1’ crashed SQAPL. Version Information added to the DLL.
3.0.23 2003-12-02	Removed multiple references of SQAPL.INI to read defaults when creating cursors – was causing poor performance in some network installations.
3.0.22 2003-08-29	Read registry in read only mode (Do not require write access to the registry).
3.0.21 2003-02-06	Look for SQAPL.INI in path where DLL loaded from if no information found in the registry.
3.0.20	A bit of cleanup
3.0.19 2001-11-19	Stop believing ODBC drivers which lie and claim that they have a timestamp column with precision less than 23.
3.0.18 2001-10-02	Development switched to Visual C++ Memory leak removed from code for fetching APL Arrays stored as SCARs
3.0.17 2001-01-16	Do not allow integers to be formatted using exponential notation when converting from Integer to Char.
3.0.16 2000-10-30	SQAX was reporting errors for the last failing row instead of the first. SQADo APL function optimised to not fetch a final empty block. SQADescribe APL function enhanced to deal with new data types.
3.0.15 2000-08-16	Added ability to open a data source in read only mode.

Appendix C: 5.0 Beta Test Change Log

Changes since version dated 2005-03-18:

- *SQAPL takes care to set all NULL values returned in ColumnWise mode to zero or empty char, some drivers were returning random values in these positions).*
- *New function SQANativeSQL added: NativeSQL no longer returned by SQADescribe (due to problems with Ingres ODBC driver).*

Changes since version dated 2005-03-13:

- *Utility function SQAParse added; converts statements with embedded bind variable declarations to "BindInfo" format.*
- *Fetch crashed if Columnwise was set to 1 and input bind variables were used.*
- *Execute of "delete from …" statement did not return the number of rows affected if none were deleted.*

Changes since version dated 2005-02-15:

- *Fixed some bugs in error/warning collection in SQAX.*
- *ColumnWise can now be used as an option on SQAConnect, to set a default for the connection.*

Changes since version dated 2005-02-03:

- *Corrected an error in the documentation regarding changes to ODBC data type numbers.*
- *Better detection of ODBC drivers which do not support "parameter sets" (SQAX).*

Changes since version dated 2005-01-27:

- *Tidying up: SQAFileVersion and superfluous comments in SQAInit gone.*
- *Bug fixed: Was impossible to use empty vector as input to VARCHAR bind vars.*

Changes since version dated 2005-01-24:

- *The version number reported by SQADescribe now has 4 elements, this version is 5.0.2.2.*
- *Conversion errors fixed: It was not possible to bind an integer value to a character variable*

Changes since version dated 2005-01-21:

- *The result returned when `SQAInit` cannot initialise translate tables is now a standard warning with -1 in the first element, followed by 10091. In earlier versions, the first element of the result was -10091.*