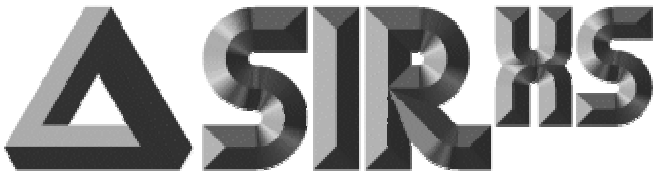


Welcome to SIR/XS

SIR/XS has many major new features including:

- Extended syntax allowing names up to 32 characters
- Non-standard names
- Enhanced standardised syntax
- STANDARD SCHEMA and STANDARD VARS
- New Common Vars definition
- New variable documentation
- New write schema options
- Upgraded record schema modification
- ADD VARS, DELETE VARS, MODIFY VARS, RENAME VARS
- Multiple Data Files
- New Batch Data Utility features including CSV Input/Output
- New Journaling and Recovery
- New XML Procedure
- New GUI Debugger
- New PQL Server
- Regular Expressions
- SEEK function to control file position
- Timestamp functions
- Extended syntax on `PROCESS CASE`
- CAT VARS in VisualPQL
- PQLForms update
- Encryption
- Enhanced date and time format specification
- Enhanced picture specification on `WRITE` and `PFORMAT`
- New GUI Controls
- IN=stdin and OUT=stdout on sirbatch execution command

SIR/XS accepts existing maintained SIR2000/SIR2002 licenses. If you are using any older version of SIR please contact us for license details.

File formats have changed significantly in SIR/XS from older versions of SIR and are not binary compatible. You need to export databases and tabfiles using your current version of SIR and import these using XS.

The menu system has been updated to reflect the new features.

Extended syntax names

Standard names in SIR/XS are from 1-32 characters, do not start with a number, are automatically capitalized and can contain letters, numbers and the four characters $, #, @ and _. For example `NAME` and `EMPLOYEE_NAME` are valid standard names. If a name is referenced in lower or mixed case, e.g. `Employee_Name` it is automatically translated to upper case.

Non-standard names can be used that do not obey these rules in some respect e.g. they contain lower case letters or special characters such as spaces. Specify a non-standard name by enclosing from 1 to 30 characters in curly braces {}. The name can contain any characters except curly braces and no translation is done on it. Non-standard names are stored without the braces and appear in sorted lists at the appropriate sort sequence. For example `{$100 dollars}` and `{Employee_Name}` are valid references to non-standard names.

The extended syntax for names applies to all named entities in SIR/XS. These are:

- Database Names
- Record Names
- Variable Names
- Index Names
- Passwords
- Tabfiles and Tables
- Families
- Members
- VisualPQL Variables
- Sub-routines
- Sub-procedures
- Labels
- Buffers

- Files (Attributes)

Non-standard names in functions

If you are using non-standard names as strings in VisualPQL functions, then the name in the string needs to be wrapped in curly brackets e.g. `VARLABSC(1,"{Employee Name}")`. The functions that pass names back to the program wrap any non-standard names in curly brackets automatically. However, please note an exception because buffer names have been allowed to have non-standard formats in earlier versions of SIR. In order to maintain compatibility with existing systems using non-standard buffer names that are enclosed in quotes, the BUFNAME function does not wrap curly brackets and the buffer references in commands continue to expect names in quotes without curly brackets.

There may be occasions when you want to create commands or other output using non-standard names and need the brackets. The new `STDNAME` function checks a name and wraps curly brackets around if it is a non-standard name.

Note: DO REPEAT; The parameter list in a DO REPEAT has always defaulted to a list of NAMES and been converted to UPPERCASE. To prevent text in a parameter list from being converted to uppercase, enclose it in delimiters using either quotes or dollar signs. If you use quotes as a delimiter, these are passed to the substitution as part of the text. If you use dollar signs as the delimiter, these are stripped out before the substitution and the case of the substitution text is preserved. If you need a non-standard name as the substitution text, specify it with curly braces as you would in a program; there is no need for additional delimiters.

GET VARS PREFIX|SUFFIX

The syntax of `GET VARS` has been extended to allow the generation of new local variable names by appending a prefix or a suffix to the original record or table variable names. e.g. `GET VARS ALL PREFIX 'EMPLOYEE_'`

SPSS

There is a new keyword `SHORTNAME` in the `SPSS SAVE FILE` procedure. This specifies that long names are truncated to eight characters for compatibility with older versions of SPSS.

Enhanced Standardised Syntax

Deprecated use of Slash /

The slash / has been used in previous versions of SIR as a separator for multiple clauses on commands. This use has been made optional for all commands. Slash still retains

meaning in various commands such as indicating a new line on a `WRITE` command, or as a delimiter on the misschar clause when the character is a blank.

Standardised Variable List Syntax

The PQL Procedures all allow variables to be specified that are to be used in the procedure, however there were some inconsistencies in the way that was done in different procedures. There is a new standard way to specify variable lists.

- The list may be a list of individual summary variable names or pairs of variable names joined with the `TO` keyword. Use of `TO` includes all variables from the first to the second.

- The procedure may output a variable using a different name. Specify this in the list either with the `AS` keyword or by following the variable name with the new name in quotes. e.g. `S(1) AS SALARY` or `S(1) 'SALARY'`

- The list may be the single keyword `ALL` to use the default procedure variables. This is essentially documentation as it has the same effect as omitting the `VARIABLES` specification completely.

- The overall list may be enclosed in brackets. If the list is enclosed in brackets, then the closing bracket indicates the end of the list. For example:

```
SPREAD SHEET  VARIABLES = (NAME GENDER SSN DIVISION)
              TITLE     = "EMPLOYEE LOCATION TABLE"
```

If the list is not enclosed in brackets then the end of the list is reached under three conditions:
There is no more input for the command.
A special character is read; slash is valid, others may be depending on the command.
A name is processed that is not a summary variable.

New Tabulate Syntax

The syntax for `TABULATE` allows new keywords to specify the individual dimensions of a table rather than this being done positionally.

Old syntax was:

```
TABULATE [[Wafer,]Stub,]Header /
                OPTIONS
```

New syntax is:

```
TABULATE HEADER = (EXP)
         STUB   = (EXP)
         WAFER  = (EXP)
                OPTIONS
```

New Schema Syntax

There are a number of new features in schema syntax.

Standard Schema.

There is a new schema command `STANDARD SCHEMA` that is similar to a `RECORD SCHEMA` command in that it signifies the start of a set of variable definitions. The set is ended with an `END SCHEMA` command. Variables are defined using a `DATA LIST` command together with any of the normal variable definition commands such as `MISSING VALUES`, `VALUE LABELS` or `VAR RANGES`. e.g.

```
STANDARD SCHEMA
DATA LIST
                POSITION                        *             (I1)
                SALARY                          *             (I2)
                SALDATE                         *
(DATE'MMIDDIYY')
VAR RANGES      POSITION                        (1 18)
                SALARY                          (600 9000)
VAR SECURITY    SALARY                          (30,30)
MISSING VALUES  POSITION
            TO  SALDATE                         (BLANK)
VALUE LABELS    POSITION                        (1)'Clerk'
                                                (2)'Secretary'
                                                 ............
VAR LABEL       POSITION                        'Position'
                SALARY                          'Salary'
                SALDATE                         'Date Salary Set'
END SCHEMA
```

Once a variable has been defined in the standard schema it can be referenced in any normal record definition with the new `STANDARD VARS` command. The benefit of this is that coding does not have to be repeated for the variable when it occurs in multiple records. Changes to the standard definition (for example updated value labels) are reflected in all the record variables that reference the standard.

Note that the extended batch data input processing definitions of `ACCEPT REC`,`REJECT REC`,`COMPUTE`,`IF` and `RECODE` are not specific to a variable and thus cannot be specified as standard and copied in.

The `STANDARD VARS` command optionally allows a variable to be renamed when used in a record e.g.

```
RECORD SCHEMA 1 EMPLOYEE
DATA LIST
            ID                                   1 -     4 (I2)
```

```
             NAME                                  6 -    30 (A25)
             GENDER                               31         (I1)
             MARSTAT                              32         (I1)
             SSN                                  33 -    43 (A11)
             BIRTHDAY                             44 -    51
(DATE'MMIDDIYY')
             EDUC                                 52         (I1)
             NDEPENDS                             53 -    54 (I1)
             CURRPOS                              55 -    56 (I1)
STANDARD VARS CURRPOS AS POSITION
```

New Common Vars Definition

Common Vars are now defined in a `RECORD SCHEMA 0 CIR` block. This can contain normal variable definition commands. e.g.

```
TASK NAME       Record Definition for CIR
RECORD SCHEMA   0  CIR
DATA LIST
                ID                                   *             (I2)
MISSING VALUES  ID                                   (BLANK)
VAR LABEL       ID                                   'Identification
Number'
END SCHEMA
```

`SIR FILE DUMP` and the batch data input utilities now support a separate input for CIR variables. That is, `SIR FILE DUMP` can write a record 0 in an appropriate format and the batch data input utilities can process that record. You can specify input format definitions for the CIR or simply allow these utilities to use default values. Note that when the batch data input utilities process a CIR, they do not expect or support the extended batch data input processing definitions of `ACCEPT REC`,`REJECT REC`,`COMPUTE`,`IF` or `RECODE`.

Record Label

The `RECORD SCHEMA` now allows you to specify a short label for the record type. The label is enclosed in quotes and is up to 78 characters. e.g.

```
RECORD SCHEMA 3 OCCUP 'Position Details'
```

The `RECDOC` function with a documentation line number of zero returns the label.

Variable Documentation

There is a new `VAR DOC` command that allows multiple lines of text to be stored on the schema to describe the variable. e.g.

```
RECORD SCHEMA   1  EMPLOYEE
VAR DOC CURRPOS The current position is a coded field
                using a copy of the standard var POSITION.
                It is the most recent permanent position of
                the employee.
```

Extended Label Lengths

Labels lengths have been standardised to 78 characters. This applies to all labels, specifically variable labels and value labels.

Sequence Commands

The `SEQUENCE CHECK` and `SEQUENCE COLS` commands are obsolete.

New Write Schema options

TO format

When SIR/XS writes a schema, by default it writes one command for multiple variables and uses the `TO` construct where a sequence of variables has the same definition. e.g.

```
MISSING VALUES  NAME
            TO  NDEPENDS                          (BLANK)
```

The `NOTO` option on `EXPORT` or `WRITE SCHEMA` suppresses the `TO` constructs.

Variable sequence format

The `VARSEQ` option on `EXPORT` or `WRITE SCHEMA` means that the schema definition for each variable is written as a set, that is all commands that apply to one variable are written, then all commands for the next variable are written, etc. e.g.

```
VAR LABEL       GENDER                        'Gender'
VAR RANGES      GENDER                        (1 2)
MISSING VALUES  GENDER                        (BLANK)
VALUE LABELS    GENDER                        (1)'Male'
                                              (2)'Female'
VAR LABEL       MARSTAT                       'Marital status'
VAR RANGES      MARSTAT                       (1 2)
MISSING VALUES  MARSTAT                       (BLANK)
VALUE LABELS    MARSTAT                       (1)'Married'
                                              (2)'Not married'
```

Common Vars repeated definitions

SIR/XS writes a new `RECORD SCHEMA 0 CIR` for any case structured database which contain the definitions for all common variables. The only reference needed for common vars in other record definitions is an entry in either the `DATA LIST` or `VARIABLE LIST` commands. Specify the keyword `COMMON` if you want complete definitions for common vars written in individual record definitions where they are referenced.

VARIABLE LIST/INPUT FORMAT

The standard style of schema output is to write variable names and input definitions as a `DATA LIST`. `VARLIST` on `EXPORT` or `WRITE SCHEMA` specifies that variable names are written as a `VARIABLE LIST` command followed by input definitions as an `INPUT FORMAT,` command. e.g.

```
VARIABLE LIST ID NAME GENDER MARSTAT ...
INPUT FORMAT (I4,T6,A25,I1,I1,...
```

Updated schema modification

The way that existing record schemas are modified has been upgraded in SIR/XS. The `MODIFY SCHEMA` command is now simply a synonym for `RECORD SCHEMA`. The `CLEAR` commands and `EDIT LABELS` commands that were specific for a `MODIFY SCHEMA` are now accepted in a `RECORD SCHEMA` and behave as before. If a schema exists and a `RECORD SCHEMA` command references it, then it is updated.

The definition of a variable can be updated with the normal set of definition commands such as `MISSING VALUES`.

Adding/Deleting Variables

Variables can be added to the schema with the new `ADD VARS` command and can be deleted from the schema with the new `DELETE VARS` command. The input position or data format of a variable can be modified with the new `MODIFY VARS` command. These three new commands are similar in syntax to the `DATA LIST` command and are used in place of it when modifying an existing schema. If a `DATA LIST` command is submitted, this completely replaces all variables in the existing definition.

The names of variables can be changed with the new `RENAME VARS` command which changes the names of existing variables while retaining all other definitions

Multiple Data Files

In SIR/XS, you can specify that a database has multiple data files, that is the .sr3 file is split into a number of files that may be in different directories.

The new `DATA FILES` command defines either a single data file or multiple data files. The command can be used to create a data file that can be in a different directory from the other database files and named something other than the database name with a `.sr3` extension.

The command can be used to create multiple data files based on key values. If the database is a case structured database, the files are primarily defined by case id values. If the database is a caseless database, the files are primarily defined by record types. For example:

```
DATA FILES  'company.s31'
            FROM (500)  'company.s32'
            FROM (1000) 'company.s33'
```

Batch Data Input

CSV

SIR/XS can process Comma Separated Variable (CSV) files as input to the batch data input utilities. There is a new keyword `CSV` on all five of the data input utilities `READ INPUT DATA`, `ADD RECORDS`, `UPDATE RECORDS`, `REPLACE RECORDS` and `EVICT RECORDS`.

The input file is a text file with values for each record in a valid CSV format. The fields must be in the correct sequence that matches the sequence of fields on the database record. A file may either contain records for a single record type, in which case, specify the record type on the utility command or may contain multiple record types, in which case the first field on each input record is the record type.

`SIR FILE DUMP` can now write CSV files and there is a new keyword `CSV` to specify this. This utility has an additional keyword `DPOINT` which specifies that explicit decimal points are written for numeric fields.

Automatic I/O columns

In previous versions of SIR, if you did not assign input/output columns to variables in a record type, they were ignored for batch data input and for file dump. These utilities have been upgraded in SIR/XS and automatically assign default columns at the end of any manually specified columns and so process these variables. This allows the use of file dump and batch data input for database maintenance tasks. There is a new keyword `NOAUTO` on the utilities to stop this processing if required.

BLANK as UNDEFINED

There is a new keyword option `BLANKUND` for `READ INPUT DATA`, `ADD RECORDS` and `REPLACE RECORDS`. `BLANKUND` specifies that blank numeric fields on the input file result in `UNDEFINED` on the record. If this option is not specified, then blanks on input for a numeric field result either in a zero value or in a missing value if a `BLANK` missing value is defined in the schema.

Processing CIR

`SIR FILE DUMP` has a new keyword `CIR` which specifies that a separate record (type 0) is written containing all common vars. The batch data input utilities recognise record type

0. If input formats have not been defined specifically for the CIR, these utilities automatically allocate columns for fixed format style processing.

New Journaling and Recovery

The journal process has been rewritten and the structure of journals and unload files is completely different in SIR/XS from previous versions. Both these files have a similar structure.

The journal file consists of a linked set of entries one entry per update run. Each entry consists of a set of images of updated records in that run. The images consist of before and after images of updated records.

Unload files can contain multiple unloads of the same database and each unload is a linked entry where the entry consists of after images of all the unloaded records.

There is a new `JOURNAL ROLLBACK` utility that removes updates and is intended for use if an update run is interrupted and does not complete properly.

When a database is connected, its status is checked to see if it was not closed properly when being updated e.g. the system 'crashed' while the database was open for update. If this is found to be the case, you are asked if you wish to automatically recover. If you choose to try to recover, a journal rollback is done.

There are new features in VisualPQL to assist in user processing of journals and unload files. The `PROCESS JOURNAL` command allows you to get information about the various entries on the file and to select one or more entries to process. When processing through an entry, data records are read in sequence from the earliest to the latest. Within the `PROCESS JOURNAL` block, a `JOURNAL RECORD IS` command starts a block that processes a specific record type. This block is given control when a record of that type is read. Within this block, you can use normal VisualPQL to access the data from the journaled record using the record variable names. viz

```
PROCESS JOURNAL
. JOURNAL RECORD IS record_type
.   PQL access to record variables
. END JOURNAL RECORD IS
END PROCESS JOURNAL
```

There is a new parameter `SIRUSER` that you can specify on start up and this is logged to the journal file whether updates are done as a single user or through Master. This can be obtained when processing the journal and it may be of interest if producing audit trails. This can be set in a VisualPQL program with the new `SIRUSER` function.

New XML Procedure

Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents. The SIR/XS XML Procedure produces a file that is an XML document. The XML File is a text file and consists of a hierarchical set of tags that enclose lower levels of tags and, at some point, enclose data. It resembles HTML, only it uses tags defined by users. Many products can now deal with XML based files. (Note. Statements in this document do not purport to describe XML or make comprehensive statements about the language but some explanation is necessary to describe the clauses on the procedure command. There are published standards for XML for those interested.)

XML has its own standard for names (which is different to SIR/XS) and any names that are generated by this procedure must meet this standard. XML names begin with alphabetic character (or underscore _) and should not start with `XML`. They are case sensitive and allow letters, numbers plus some special characters but no spaces.

An example bit of XML from within a document might be:

```
<company>
   <person>
       <name>John D Jones</name>
       <salary>2150</salary>
       <birthday>1986</birthday>
   </person>
   <person>
       <name>James A Arblaster</name>
       <salary>1500</salary>
       <birthday>1981</birthday>
   </person>
</company>
```

`XML SAVE FILE` Procedure

The procedure has a number of standard clauses, common to most procedures:

```
FILENAME = filename
BOOLEAN = (logical expression)
MISSCHAR = character
SAMPLE = fraction
SORT   = variable,....
```

It also has a number of specific clauses

```
ROOT       = 'string'
BREAK      = break_variable (TAG = 'string',
                             ATTRIBUTES = (varname (format) ),...),
ELEMENTS   = (varname (format) ),...)),
             break_variable ..........
DTD        [= filename]
SCHEMA     [= filename]
```

The XML file consists of a well formed hierarchy and the `ROOT` is the top-level outermost component of this. This defaults to `SIR_XS_ROOT` if not specified. Specify a valid XML name as the root that the processing application expects.

The `BREAK` clause must be specified and determines the hierarchy of the XML document. Each variable listed on the clause means one further level of nesting. The first variable is the outer level. By default the variable name is used as the tag. Specify a `TAG =` to override this.
There are two ways in which data can be included in a hierarchical level. You can specify individual data `ELEMENTS` or a set of `ATTRIBUTES`. Both of these name data variables but they appear in a different way in the output. Elements appear as individually tagged items whereas attributes appear within the start tag. For example, if there are three data variables for a person `Name, Salary, Birthday` then using elements, the output looks like:

```
<person>
<NAME>John D Jones</NAME>
<SALARY>2150</SALARY>
<BIRTHDAY>01 15 78</BIRTHDAY>
</person>
<person>
<NAME>James A Arblaster</NAME>
<SALARY>2650</SALARY>
<BIRTHDAY>12 07 82</BIRTHDAY>
</person>
```

Using attributes the output looks like:

```
<person NAME="John D Jones" SALARY="2150" BIRTHDAY="01 15 78">
</person>
<person NAME="James A Arblaster" SALARY="2650" BIRTHDAY="12 07 82">
</person>
```

If designing an XML application from scratch, this may be a matter of style and choice. If supplying a file to an existing application, then it is a matter of matching a specification.

The name of the element or attribute is the variable name. If this does not match the tag required, you can alter this as per the standard method for specifying variable lists in procedures (e.g. `S(1) AS SALARY` or `S(1) 'SALARY'`). Specify any formatting to be applied to the data as per the normal formats specified on a `WRITE` command.

One or two additional files may be produced that describe the XML file written. You can specify a DocumentType Definition or DTD. XML provides an application independent way of sharing data. With a DTD, independent groups of people can agree to use a common DTD for interchanging data. Your application can use a standard DTD to verify that data that you receive from the outside world is valid. You can also use a DTD to verify your own data. If you specify the `DTD` keyword, the default filename is the name of the main XML file produced with the extension .dtd

XML Schema is an XML based alternative to DTD. You can also produce an XSD file that describes the XML. If you specify the `SCHEMA` keyword, the default filename is the name of the main XML file produced with the extension .xsd.
Specifying either Schema or DTD changes the header information written to the main

XML file and so informs other processes that a descriptive file exists. You may find other applications that process the XML need a certain style of descriptive file.

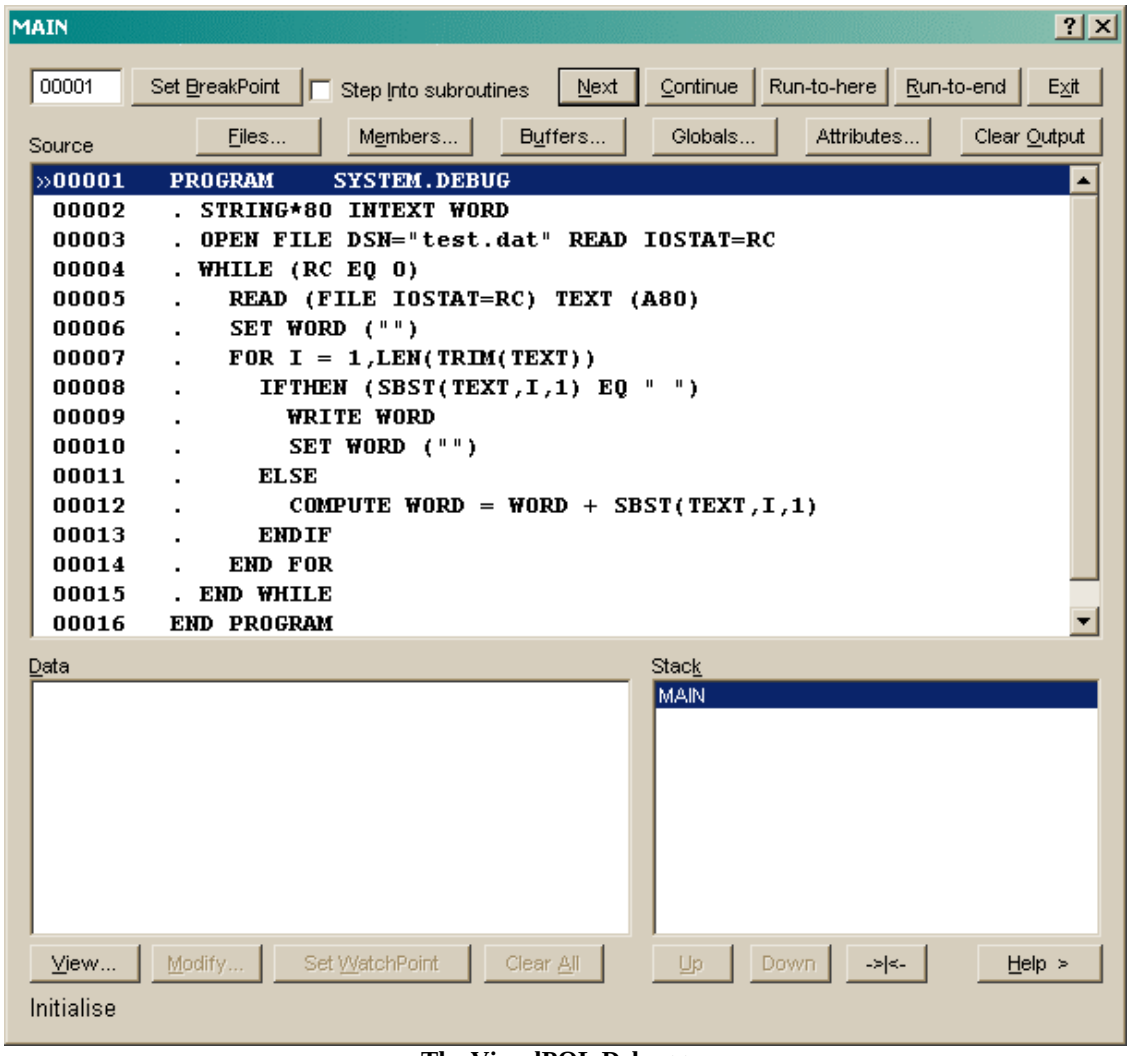
Example XML procedure

```
RETRIEVAL /PROGRESS
. PROCESS CASES ALL
.   get vars id
.   PROCESS RECORD EMPLOYEE
.     GET VARS NAME BIRTHDAY SALARY
.     PERFORM PROCS
.   END PROCESS RECORD
. END PROCESS CASES
XML  SAVE FILE
     FILENAME = "c:\sirxs\alpha\XML2.XML"
     ROOT = 'company'
     BREAK = ID (TAG = 'person' ATTRIBUTES = (name salary birthday))
     SORT  = ID
     schema
END RETRIEVAL
```

New PQL GUI Debugger

The old execution window debugger is obsolete and there is a new GUI debugger.
The `DEBUG` clause on `PROGRAM/RETRIEVAL/SUBROUTINE` is altered. The words `MONITOR` or `FULLMONITOR` are obsolete. The clause is now:
`[DEBUG [= name]]`
This causes a debug version of the module to be stored as a subroutine on the procedure file. If you are compiling a real subroutine, the name need not be specified. If the name is not specified on a `PROGRAM` or `RETRIEVAL`, it defaults to `SYSTEM.DEBUG` To debug a program, start the GUI debugger from the menu system. This allows you to select a module for debugging and provides various windows to list source code and current position, to set breakpoints, to list variables, inspect and alter values and to set watchpoints and to step through modules, into and out of subroutines.

To start the debugger, first compile a program with the DEBUG keyword then select **Debug...** from the **Program** menu. Select the compiled object (**`SYSTEM.DEBUG:O`** is the default) from the member list.



The VisualPQL Debugger

Please see the VisualPQL GUI Debugger for instructions on debugging a routine.

New PQLServer

The SIR/XS PQLServer is a new executable that allows another standard SIR/XS session to connect as a client and to transmit commands to the server, execute those commands remotely and retrieve output. This is done with a set of PQL functions. The PQLServer must be started to enable clients to communicate to it across the network. The client processes do not require any access to files or databases that are local to the server and the two processes (client/server) may be using different hardware/operating systems e.g. client on windows, server on Unix.

From the client point of view processing is as follows:

- Client logs on to server and establishes a connection with the server that is is used in all subsequent server functions.

- Client sends any number of lines of text that would include SIR commands.
- Client starts execution of previously sent commands. Any settings or output from a previous execution from the same client are reinitialised, the commands are run and a completion code is returned at which point any output is waiting on the server. Commands can include all SIR commands and can use procedures, etc. Note that commands must include connecting any databases/tabfiles/procedure files needed each time commands are submitted and executed. There are no saved settings between executions. The process may read/write files, update databases and generally do anything that a batch run of SIR could do. While executing, no communication is happening with other possible clients. It is thus good practice to keep execution streams as short as possible.
- The client may choose to wait for the execution to finish or to carry on processing locally and subsequently test to find if the execution has completed successfully.
- Client gets count of number of lines of output and can then get each text line or skip over unwanted lines. The server transmits lines in groups. If skipping lines and the lines have not yet been transmitted, they are skipped on the server. Lines once returned or skipped are no longer available. The client can get a count of the number of lines available at any point.
- Client can repeat the process.
- Client logs off when finished.

e.g.

```
program
compute rc = serlog ('TONYDELL:4000','')
write rc
compute x = sersend ('PROGRAM')
compute x = sersend ('WRITE "HELLO WORLD"')
compute x = sersend ('END PROGRAM')
compute rc = serexec (1)
write 'rc = ' rc
compute olines = serlines(x)
write 'lines ' olines
for i=1,olines
. compute line = serget (0)
. write line
rof
compute client = serlog ('','')
end program
```

Client Functions when using PQLServer

`SERADMIN` Various server administration capabilities (returning numeric values)

`SERADMIS` Various server administration capabilities (returning string values)

`SEREXEC` Instructs server to execute previously sent commands

`SERGET` Gets a line of output from server

`SERLINES` Asks server how many lines of output are left

`SERLOG` Logs on to the server

`SERSEND` Sends a string to the server

`SERSENDB` Sends a buffer to the server

`SERTEST` Asks server if execution has completed

PQLServer Functions

(These have no effect if used in a program that is not running on the server)
`SERNOOUT` Suppresses server output

`SERWRITE` Writes a line of output from server

New PQLServer commands

```
CLEAR SERVER NOOUTPUT
SET SERVER NOOUTPUT
```
These can be used anywhere in the command stream to selectively control what output is made available for retrieval by the client. SET means that any output directed to standard output is thrown away. If this command is embedded in a PQL program, it affects any compilation listing. Use the `SERNOOUT(n)` function for execution time control.

Regular Expressions

There are two new functions that allow you to manipulate strings with *Regular Expressions*. A regular expression is one where symbols describe the matching that is required. The meaning of the symbols needs to be specified and there is a standard for regular expressions used by many packages. SIR has had its own regular expression processor and these functions allow you to choose whether to use PERL standard expressions, POSIX standard expressions or SIR expressions.

`REGEXP` searches a string using a regular expression and returns whether the nth occurrence of the string has been found and the position in the string of the start of the match.

`REGREP` takes a string and two regular expressions and replaces matches from the first expression with text specified by the second.

SEEK function

The new `SEEK` function sets a position on an open file.

Timestamp functions

There are new timestamp functions. `DTTOTS` takes a date and time integer and returns a timestamp. A timestamp is a real*8 representation and is the number of seconds since the start of the SIR/XS calendar. You can do calculations between timestamps and the individual date and time components can be extracted using the `TSTODT` and `TSTOTM` functions before using any other date and time functions e.g. for print formatting.

Extended syntax on PROCESS CASE

The `COUNT` and `SAMPLE` options on `PROCESS CASE` now allow variables and expressions as parameters rather than requiring explicit integer variables.

CAT VARS in VisualPQL

Local categorical variables can now be defined in VisualPQL.

PQLForms Update

There is a new clause `TITLE` on the `SCREEN` command. This allows you to specify PQL that constructs screen titles on single and multi-page screens.

There is a new sub-clause `FONT` as part of the PQLForms general clauses that allows specification of non-standard fonts including color. This has also been implemented in the forms screen painter allowing easy use of fonts.

The `IF` sub-clause has been implemented for `FBUTTON` which allows the button to be enabled or disabled (greyed out) according to specific conditions.

Encryption

There is a new schema command `ENCRYPT [ON |OFF]`

`ENCRYPT` turns on data encryption for this database. This means that all data records in the database are encrypted on disk and are thus protected against scrutiny from software other than SIR/XS. The encryption method used is a version of the publicly available Blowfish algorithm using a 256 bit key.

All data records are encrypted, however keys in index blocks are held in unencrypted format. Do not use names or other recognisable strings as keys if this data is sensitive and requires protection. Unloads and journals for encrypted databases are themselves encrypted. Text files are all unencrypted. Schemas and procedures are unencrypted.

`ENCRYPT OFF` turns encryption off for a database. Encryption can be turned on and off without ill effect. Records are written according to the current setting; records are read and recognized as to whether they require decryption.

Passwords and security levels are encrypted on all databases. There are encryption/decryption functions in VisualPQL if users need to encrypt data for themselves but these use a user specified key - the SIR/XS system key is used for database encryption.

`CRYPTKEY` Sets the key for the encryption functions.
`DECRYPT` Decrypts an encrypted string.
`ENCRYPT` Encrypts a string.

Enhanced date and time format specification

Variables can now be defined with extended date and time formats. These formats allow output formats which retain specific separators such as '/' and allow days of the week and 12 hour times to be specified. See date and time formats for a complete description.

Enhanced picture specifications

Numeric variables with an output specification on the `WRITE` command or formatted by `PFORMAT` have a picture specification that describes the required format. This now has additional capabilities for floating dollar sign and negative numbers. The documentation has been upgraded to cover all picture specification features.

Improved VARMAP summary

The summary information produced by the VARMAP keyword on RETRIEVAL/PROGRAM has been improved to make it more readable.

New GUI Controls

Tree Control

A tree control can display a hierarchical list of items such as the structure of a database or the data within that database. The items in the tree list can have child items and, if they do, then they have a button that shows or hides the child items.

Slider Control

A *slider* or *trackbar* control displays a scale with a movable knob. If the height is greater that the width then the slider is drawn vertically, otherwise it is drawn horizontally.

Spin Control

A *Spin* or *Up/Down* control displays a numeric edit box with a pair of arrows. The up arrow increments the value in the edit control and the down arrow decrements it.

Progress Control

A *Progress* control displays a read only progress meter. The meter can be horizontal or vertical depending on the height and width settings.

ComboBox Control

A *ComboBox* control is a combination of an edit and a choice control and can use the same functions and commands as these controls.

Other New GUI Features

Images on buttons

Buttons can now have images rather than text. Define a button as per standard buttons and then use the `SET IMAGE` command to specify a bitmap to use for the button.

Images can optionally be centred or resized to fit the image control.

Functions

`SETRANGE` Sets the minimum and maximum possible values for the a slider, spin or progress control.

`BRANCH` Adds a new node to a tree. The node is added as a child node of the parent.

`BRANCHD` Deletes the node from a tree.

`NBRANCH` Returns the number of child nodes of the given node.

`BRANCHN` Returns the node number of the nth child nodes of the given node.

`SCROLLAT` Gets a position in a gui scrollable item

`SCROLLTO` Sets a position in a gui scrollable item

`CLIPAPP` Adds text to the clipboard.
`CLIPGET` Gets text from the clipboard.
`CLIPLINE` Gets count of lines in the clipboard.
`CLIPSET` Clears the clipboard and adds text to the clipboard.

Messages

`message VSCROLL m_id, m_arg1`
`m_id` contains the id of the list or text control that has been scrolled;
`m_arg1` contains the position of the item that is the top most *visible* in the list.
This message could be used to syncronise the scrolling of two related lists.

`message RMOUSE m_id, m_arg1, m_arg2`
`m_id` contains the id of the control where the right mouse button was clicked. If the mouse was not over a control then -1 is sent.
The information in `m_arg1` and `m_arg2` depends on the type of control:
List controls send the position of the item in the list and a double click indicator.
Otherwise the x,y coordinate relative to the top left of the control is returned.

Interface Enhancements

There have been many enhancements to the default menu system and dialogs. These include:

- Larger lists, both to take account of longer names and for easier readability;
- Search Help allows you to search the help system for particular words that may not necessarily be indexed;
- More Procedures... under the procedures menu lets you store and document commonly used programs. There are some "system" procedures provided. User Procedures can be stored with a specific database or on a standalone procedure file.
- Data Entry menu item starts a default form or can be set to start any form (or program) using the Data/Forms... dialog.
- More preference settings including the maximum height of the dropdown list in choice/combo controls; the size of the internal editor, dbms command and debug dialogs;
- Enhancements to the dialog/forms painter including fast page navigation with PgUp PgDn Home and End; WYSIWYG font attributes.

Other New Features

The default length of a string variable has changed from 20 to 32 characters.

Several values returned by the SYSTEM function have become obsolete and replaced with new codes:

- SYSTEM(3) now returns the update level of the current record (was not used);
- SYSTEM(44) returns DB encryption indicator (was Template Storage);
- SYSTEM(54) returns the number of data files defined (was Message Level);
- SYSTEM(79) lines in def member (was Execution Window rows);
- SYSTEM(80) window paging on / off (was Execution Window columns);

Automatic disconnection of idle master clients. The new SETAKL function (Set AutoKill Limit) allows setting of a time limit for clients of master. If a client is idle for the given number of minutes then they will be automatically disconnected.

```
COMPUTE RC = SETAKL(minutes,password)
```

Compatibility Issues

If you have existing SIR systems, export your existing databases and tabfiles and import into the new version. You should make sure your production applications operate correctly on any new version of the software before stopping use of existing versions.

SIR/XS databases are *NOT* backward compatible with SIR2002 and earlier but can be exported and re-imported if needed to be used with older versions but the OLD option must be used with the export to avoid creating new schema constructs that are incompatible with earlier versions. (Note. Using specific SIR/XS features such as non-standard names may result in incompatibilities anyway.)

Since SIR/XS now supports non-standard names using {} as delimiters, the use of double quotes in SQL as delimiters for non-standard names is not recommended but is still supported.

Old text style screens are no longer supported in DBMS/VisualPQL. The following text style screen commands are no longer supported:

```
ACCEPT CHARACTER
BELL
BOX
CURSOR
CONNECT TEMPLATE
CREATE TEMPLATE
DELETE TEMPLATE
ERASE TEMPLATE
DISPLAY TEXT
EXPORT TEMPLATE
ERASE SCREEN
FIELD INPUT
FILL
HORIZONTAL MENU
KEYPAD ON
LINE ATTRIBUTE
LINE CHARACTER
MAPKEY
MODIFY TEMPLATE
MOUSE
POP TEMPLATE
PUSH TEMPLATE
PURGE TEMPLATE
REFRESH SCREEN
RESTORE TEMPLATE
SCREEN GRAPHICS
```

```
SCROLL TEMPLATE
SENSE MOUSE
SOUND function
STORE TEMPLATE
VERTICAL MENU
```

Forms is still supported although it is not being developed further and does not have constructs to use any database secondary indexes. Forms is now the only component of SIR/XS to use a text style full screen.

The default screen font used in SIR/XS is different to that used in 2002 and this may cause some size differences in dialogs. The font used to be MS Sans Serif but this is no longer included by default in windowsXP and this font doesn't have all the characters available in other fonts (eg "…"). The default font is now Arial. To alter the default font, set sir.fnamm to a fontname either manually by editing the sir.ini file or with `UPSET` in the start menu. e.g.

```
UPSET("sir.fnamm","MS Sans Serif")
```

This sets the font to be the same as used in SIR2002.

IN=stdin and OUT=stdout on sirbatch execution command

When running sirbatch you can use stdin and stdout as the filenames for the IN= and OUT= parameters.
IN=stdin takes input from the standard input stream. This could be the keyboard, a pipe or redirected input using "<". The `PROMPT` execution parameter will cause a prompt to be written to the screen each time an input line is read.
OUT=stdout sends output to the standard output. This could be the screen, a pipe to another command or redirected output using ">".