SCULPTOR 2.5 & 2.5a RELEASE NOTES

Introduction

Sculptor version 2.5 is a consolidation of all previous 2.4 versions, together with enhancements to the Sculptor run-time system, particularly in respect of support for dates.

Version 2.5a introduces a new file access command **wind**, an improved version of **kfcheck** and an extra utility for generating simple screen form reports.

Users of these notes should check the Sculptor version number given on the supplied media to determine if their system is version 2.5 or 2.5a.

contents

For questions about these release notes, please contact Sculptor Technical Support.

Sculptor Ltd, Bournehall House, Bournehall Road, Bushey, Watford, WD2 3EH.

telephone: +44 (0) 181 420 4701 facsimile: + 44 (0) 181 421 8081

email: 100561,2055@compuserve.com

SCULPTOR 2.5 & 2.5a RELEASE NOTES

Version 2.5

The enhancements in version 2.5 are entirely in the run-time system. In most cases, existing compiled Sculptor programs do not need to be recompiled to use the new features; the one exception to this rule is explained in the *points to note* section of these release notes.

A new **lcf** string has been added to **sage** and **sagerep**. It is important that users of **lcf** be careful not to confuse the new string, which is the second item in the list, with the decimal point configuration string, which has now moved to third place in the list.

Date support

As the millennium approaches, software users and developers are increasingly concerned about the issue of data entry and storage of dates with an ambiguous year component—does a date such as 1/1/1 mean 1/1/1901 or 1/1/2001 or even 1/1/1801? These notes will refer to a date with a 1 or 2 digit year component (i.e. a date with no specific century) as an ambiguous date.

There has never been any ambiguity in the storage of the Sculptor date type. In any version of Sculptor, dates are stored, retrieved and manipulated with a full 4 digit year. Ambiguity can arise, however, in keyboard data entry.

In summary, the features provided by Sculptor 2.5 to deal with ambiguous dates entered from the keyboard are:

1. Application of a configurable algorithm to deal with ambiguous dates. The default configuration calculates, for example, that 1/1/1 is a short form of 1/1/2001, whereas previous versions of Sculptor simply assume that all ambiguous dates are in the twentieth century.
2. If a date is formatted with a full four digit year (e. g. dd/mm/yyyy,)it is possible to configure the system so that users are forced to enter all four digits.
3. If a run time system is configured to allow ambiguous dates and the user enters such a date, **sage** will redisplay it with a full four digit year when the user presses the return key, *provided the date format allows four digits in the year field.*

Using **lcf** *to configure the ambiguous date handler*

All the configurations outlined above are controlled using **lcf**. The use of **lcf** is described in the main Sculptor Reference Manual.

sage and **sagerep** both have a new **lcf** string which controls the ambiguous date handler. The new string is the second **lcf** string, coming after the standard default date format. The format of the new **lcf** string is:

```
    integer [,Error String ]
```

The integer, known as the century break point, is used in determining the century to which ambiguous dates belong.

The optional error string contains a message to be output when a date with a four digit year format is input from the keyboard with less than four digits. If no error string is given one of the date algorithms described below will be applied to resolve any ambiguity. Completely empty dates (i.e. all spaces) are always permitted.

If the breakpoint is any negative number, the century of an ambiguous date is simply set to that of the current system date.

If the break point is a 4 digit number it is known as an absolute break point, if it is 1 or 2 digit number it is known as a relative break point.

Absolute break points

The absolute break point is used to set a year, normally in the twentieth century, to define which range of 2 digit years belong to the twentieth century and which to the twenty first.

example:

Assume that the **lcf** string is set to

```
1950,Please enter all four year digits
```

This sets an absolute breakpoint—absolute because it has four digits. Two digit years less the 50 are taken to be in the twenty-first century, those greater than or equal to 50 are taken to be in the twentieth century.

The error string is set, which means that ambiguous entry is allowed for dates which are formatted ambiguously (e.g. dd/mm/yy) but not for dates which specify all four year digits.

The following table shows example input dates against dates understood by the runtime system:-

User input	*Date understood by the system*
1/1/0	1/1/2000
1/1/1	1/1/2001
...	...
1/1/49	1/1/2049
1/1/50	1/1/1950
1/1/51	1/1/1951
...	...
1/1/99	1/1/1999

Users of specialist applications, such as historical archives, may find a use for absolute break points in centuries other than the twentieth. The general principle is the same as in the above example. The first 2 digits are known as the base century and the second 2 digits the break-point. If a 2 digit year, yy, is entered, the century is worked out according to the scheme:

```
if yy < break point
    century = base century + 1
else
    century = base century
```

Relative break points

Using a relative break point, which has a two digit entry in the **lcf** string, is a more subtle way to deal with ambiguous dates.

A relative break point is a point set to a certain number of years relative to the current date. Sculptor 2.5 is shipped with a relative break point of 10. For a system running in 1997 this sets the actual break point to 1997 + 10 = 2007. 2 digit years in the range 0-7 are calculated to be in the range 2000-2007, all other years (i.e. the range 8-99) are calculated to be in the range 1908-1999.

When the operating system date moves on to 1998 the actual break point moves with it by one year and these ranges become 2000-2008 and 1909-1999.

example:

Set the **lcf** string to

```
10
```

There is no error string, therefore ambiguous dates are allowed even when the format specifies 4 digits in the year component. Assume the current year, as understood by the operating system is, 1997:

```
Current Year = 1997
Relative break point = 10
```

User input	*Date understood by the run-time system*
1/1/0	1/1/2000
1/1/1	1/1/2001
...	...
1/1/7	1/1/2007
1/1/8	1/1/1908
1/1/9	1/1/1909
...	...
1/1/99	1/1/1999

As the operating system date changes the date understood by the run-time system changes with it. In 1998 the actual break-point becomes 1998 + 10 = 2008 and the table becomes:

```
Current Year = 1998
Relative break point = 10
```

1/1/0	1/1/2000
1/1/1	1/1/2001
...	...
1/1/7	1/1/2007
1/1/8	1/1/2008
1/1/9	1/1/1909
...	...
1/1/99	1/1/1999

In 2000 the table becomes:

```
Current Year = 2000
Relative break point = 10
```

1/1/0	1/1/2000
1/1/1	1/1/2001
...	...
1/1/10	1/1/2010
1/1/11	1/1/1911
1/1/12	1/1/1912
...	...
1/1/99	1/1/1999

Setting a relative break point of 0 will ensure that all 2 digit years are calculated to be in the past or present, never the future:

```
Current Year = 1997
Relative break point = 0
```

1/1/0	1/1/1900
1/1/1	1/1/1901
...	...
1/1/97	1/1/1997
...	...
1/1/98	1/1/1898
1/1/99	1/1/1899

Setting a relative break point of 99 will en-sure that all 2 digit years are calculated to be in the future or present, never the past:

```
Current Year = 1997
Relative break point = 99
```

1/1/0	1/1/2000
1/1/1	1/1/2001
...	...
1/1/97	1/1/1997
1/1/98	1/1/1998
1/1/99	1/1/1999

In more formal terms, the relative breakpoint algorithm may be expressed thus:

1. Let the ambiguous year for which we wish to calculate the century be `ambiguous_year`.
2. Let the relative break point be `relative_break`.
3. Add `relative_break` to the current system year. The resulting number will have a century and a two digit year. Call these quantities `test_century` and `test_year`.
4. Now work out the century of `ambiguous_year` by the formula

if `ambiguous_year` is greater than or equal to zero and `ambiguous_year` is less than or equal to the `test_year`, then the required century is `test_century`. Otherwise it is `test_century -1`.

example

```
ambiguous_year = 3
relative_break = 10
current system_year = 1997
add 1997+10 to get 2007, then
test_century = 20
test_year = 7
```

since 3 is greater than or equal to 0, and 3 is less than or equal to 7, the required year is equal to `test_century` (i.e. 20)

Points to note

1. The problem of ambiguous dates can be eliminated completely if dates are formatted with a full four digit year component and the Sculptor run time system is configured to force data entry to conform to the format.
2. All the new features in Sculptor 2.5 are incorporated into the run-time system, there is therefore no need to re-compile any program unless existing dates which are ambiguously formatted as dd/mm/yy are to make use of the new feature of forcing unambiguous dates. In this situation there is not enough room in the date format to type in all the required digits, therefore the date field must be formatted as dd/mm/yyyy and the programs which use it re-compiled.
3. Everything said in these notes in respect of dates formatted in the European styles dd/mm/yyyy and dd/mm/yy applies to the equivalent American formats mm/dd/yyyy and mm/dd/yy.
4. The date configuration string in **lcf** must always have a breakpoint even if the error string is set because the Sculptor run-time system must always have a way to deal with deal with ambiguous dates. If the **lcf** string contains rubbish, i.e. does not conform to the strict syntax of **integer [,error string]**, the century of an ambiguous date will be set to that of the system year and the error string will not be set.
5. String literals in programs which contain dates and strings input with the **get** statement are dealt with precisely as if they had been entered at the keyboard. It is very bad practice to use ambiguous dates in string literals. For example:
   ```
   !temp x ,, d4
        x = "1/1/1"
   ```
   The actual value assigned to the variable **x** is completely determined by the default setting of the **sage** and **sagerep lcf** string, which is a relative breakpoint of 10.

Best practice for Sculptor 4GL programs and SQL scripts is not to use ambiguous years at all.

Handling of riu *traps in sage*

When a screen form program encounters a locked record and the default **riu trap** is used to handle the situation, **sage** will output the standard message for the condition together with the name of the file upon which the lock is applied. In the event that this will not fit into the standard screen width **sage** will work as it did prior to version 2.5 (i.e. output the standard waiting message only).

SCULPTOR 2.5 & 2.5a RELEASE NOTES

Version 2.5a

New features

The **wind** command

SYNTAX

wind *file_id* [**index=** *index_name*]

The **wind** command is analogous to the rewind command in that it will position the file pointer past the end of the file or alternate index so that a subsequent prev command will retrieve the last record in the file for which the key is not all binary 1's.

In all practical applications, this yields the last record in a file.

Like **rewind**, which does the same but with binary 0's, the **nrs** condition is true after the command has been called.

Unlike **rewind**, there is no support for sequential files.

Improved **kfcheck**

kfcheck has been updated to make it more efficient in terms of its speed and resource requirements.

If **kfcheck** finds file size errors, such as a data file which is not a multiple of its record length, it will attempt to fix the problem before continuing. This will occur for all levels of **kfcheck**.

The levels of kfcheck now run from 0 - 4:-

0	Check file lengths and all index header records
1	+ Check integrity of main index
2	+ Check integrity of secondary indexes
3	+ Check data file and secondary index key counts
4	+ Read and check keys in secondary indexes

kfcheck level 5 is now obsolete but users with scripts which use a level 5 **kfcheck** will not need to change them as a level 5 **kfcheck** request defaults to level 4. Level 3 is adequate for all practical purposes. Only in the event of system failure should level 4 be used. For backward compatibility, level 4 is the default.

The diagnostics output by **kfcheck** have been improved. In most cases the new diagnostic messages will not be of use to end users but will assist Sculptor Technical support personnel. The new messages are stored in a new version of **kfcheck.eng**. It is therefore important when installing Sculptor that the normal installation procedure is not by-passed as this could result in a wrong version of **kfcheck.eng** being used and spurious messages being output.

oldsg/oldrg & **sg/rg**

Versions of Sculptor prior to version 2.4 were shipped with simple screen and form generator programs. These programs have now been restored and are now known as **oldsg** and **oldrg**.

We would recommend that the new versions of **sg** and **rg** be used as file browsers, but the old versions (**oldsg** and **oldrg**) be used as starter code for user applications.