Addendum

for Updates to JAM Release 5.03

for PL/1

Part Number R332–00A

August 3, 1992

Note of Explanation

This addendum describes new features in release 5.03 of **JAM**. This addendum is for the PL/1 *Programmer's Guide*. There are separate addenda for Volumes 1 and 2 of the **JAM** 5.03 documentation set.

Several insertion pages (or A–pages) are included for new library routines and utilities in **JAM** 5.03. These pages should be inserted into your **JAM** *Programmer's Guide* and *Utilities Guide* at the appropriate location. For example, page A–195 should be inserted before page 195.

Note that the page numbers for the *Utilities Guide* refer to the August 1, 1991 printing of the **JAM** manual. Page numbers in the *Programmer's Guide* refer to the March 1, 1991 printing of the PL/1 *Programmer's Guide*.

PL/1 Programmer's Guide

Page 92: New Behavior and Return Codes for `xsm_ascroll`

The library routine `xsm_ascroll` takes as arguments a field number and an occurrence. It scrolls an array such that the requested occurrence is in the specified field. If the requested occurrence cannot be placed in the specified field because it is one of the first or last occurrences in a non–circular array, then `xsm_ascroll` scrolls the occurrence onto the screen and returns the occurrence number of the occurrence that is actually in the specified field.

Page 168: Inquiring Help Level via `xsm_inquire`

The global variable `I_INHELP` now contains the level of help that the user is in, instead of just a true/false value. There may be up to five levels of help. Use `sm_inquire` to query the value of this variable. A return of zero indicates that the user is not in help, a return of 1 through 5 indicates which help level the user is in.

Page 194: `xsm_keyoption`

Certain keys can not be translated via the `KEY_XLATE` argument to `sm_keyoption`. These are: INS, REFR, SFTS, LP, and ABORT. They may, however, be disabled via the `KEY_ROUTING` argument, or intercepted via a keychange function

Page 246: Percent Escapes in `xsm_query_msg`

Percent escapes are now supported for controlling the attributes of query messages. The sequences are the same as those for `xsm_emsg`, and detailed on page 214. Note that `%Mu` and `%Md` are not supported. Query messages from JPL can also now use percent escapes.

Page 292: MDT bits and Scrolling Arrays

When lines are inserted or deleted from scrolling arrays via INSL or DELL, the MDT bits for all occurrences after the insertion or deletion are no longer set. In a database application, this prevents the need for unnecessary processing to write potentially large amounts data that have not changed. For large arrays, it can save a significant amount of processing time.

bin2pl1

convert binary **JAM** files PL1 declare data

SYNOPSIS

```
bin2pl1 [-fv] PL1-file binary-file...
```

OPTIONS

`-f` Overwrite an existing output file.

`-v` Generate list of files processed.

DESCRIPTION

This program converts binary files created with other **JAM** utilities into PL1 source. ***PL1-file*** is usually a new file name. (To overwrite an existing file, you must use the `-f` option.)

When the utility creates the PL1 source file, it generates a data file for each of the binary input files. The name of the data file is derived from the binary file name, with the path and extension removed, and given the extension `.incl.pl1`.

The application program should include the data file in the program that uses it. The `_d` variants of certain library routines (`d_window`, `d_form`, `d_at_cur`, `d_keyset`, `d_msg_line`) can then be used.

`bin2pl1` output files may be compiled, linked with your application, and added to the memory–resident form list. (See the **JAM** *Programmer's Guide* for more information on memory–resident lists.) The following files may be made memory–resident:

- key translation files (`key2bin`)
- setup variable files (`var2bin`)
- video configuration files (`vid2bin`)
- message files (`msg2bin`)
- JPL files (`jpl2bin`)
- screen files (`jxform`)

There is no utility to convert ***ascii-file*** back to its original binary form after using `bin2pl1`. **JAM** provides other utilities that permit two–way conversions between binary and ASCII formats. For screens, these utilities are `bin2hex` and `f2asc`.

ERRORS

`Insufficient memory available.`
Cause: The utility could not allocate enough memory for its needs.
Corrective action: Try to increase the amount of available memory.

`File "%s" already exists; use '-f' to overwrite.`
Cause: You have specified an output file that already exists.
Corrective action: Use the –f flag to overwrite the file, or use another name.

`"%s": Permission denied.`
Cause: An input file was not readable, or an output file was not writeable.
Corrective action: Check the permissions of the file in question.

copyarray

copy the contents of one array to another

SYNOPSIS

```
declare destination_fld    fixed binary(31);
declare source_fld         fixed binary(31);
declare status             fixed binary(31);
status = xsm_copyarray(destination_fld, source_fld);
```

DESCRIPTION

This routine copies the contents of the array containing `source_fld` into the array containing `destination_fld`. `source_fld` and `destination_fld` are field numbers. They may be the field number of any of element in the respective array.

The developer is responsible for insuring that the arrays are compatible. Data in source array occurrences that are too long for the destination array are truncated without warning. Data in source array occurrences that are shorter than the destination array's field length are blank filled (with respect for justification).

If the source array has more occurrences than the destination array, the data in the extra occurrences are discarded. If the source array has fewer occurrences than the destination array, trailing occurrences in the destination array are cleared of data (but not de-allocated).

`copyarray` sets the MDT bit and clears the VALIDED bit for each destination array occurrence, indicating that the occurrence has been modified and requires validation.

The variant, `xsm_n_copyarray`, searches the LDB for either array if the named field is not found on the screen. However, if the destination LDB item has a scope of 1, meaning that it is a constant, then it is not altered and the function returns –1.

RETURNS

–1 if either field is not found or if the destination array in the LDB has a scope of 1.
0 otherwise.

VARIANTS

```
status = xsm_n_copyarray(destination_name, source_name);
```

RELATED FUNCTIONS

```
status = xsm_clear_array(field_number);
length = xsm_getfield(buffer, field_number);
status = xsm_putfield(field_number, data);
```

next_sync

find next synchronized array

SYNOPSIS

```
declare field_number        fixed binary(31);
declare next_array          fixed binary(31);
next_array = xsm_next_sync(field_number);
```

DESCRIPTION

Given a field number, this function finds the next array synchronized with the given field, and returns the field number of the corresponding element in that array. The next synchronized array is defined as the one to the right. If `field_number` is in the rightmost synchronized array, the function returns the corresponding element in the leftmost synchronized array (ie– it wraps around the screen).

RETURNS

The field number of the corresponding element in the next synchronized array if there is one.
Otherwise, the field number the function was passed.

soption

set a string option

SYNOPSIS

```
declare option              fixed binary(31);
declare newval              char(256) varying;
declare oldval              char(256) varying;
oldval = xsm_soption(option, newval);
```

DESCRIPTION

Use `xsm_soption` to alter during run-time the default string options defined in `smsetup.incl.pl1`. The following table lists the valid mnemonics for `option`:

Mnemonic	*Description*
SO_EDITOR	Editor to use in JPL windows.
SO_FEXTENSION	Screen file extension.
SO_LPRINT	Operating system print command.
SO_PATH	Search path for screens and JPL procedures.

These variables are fully documented in the **JAM** *Configuration Guide*, under "System Environment and Setup Files."

RETURNS

The old value for the specified option.
0 if the option is invalid or a malloc error occurred.

RELATED FUNCTIONS

```
oldval = xsm_option(option, newval);
```

wrotate

rotate the display of sibling windows

SYNOPSIS

```
declare step                fixed binary(31);
declare status              fixed binary(31);
status = xsm_wrotate(step);
```

DESCRIPTION

If two or more sibling windows are on the top of the display, this function may be used to rotate the sequence of the sibling windows. `step` is a positive or negative integer equalling the number of screen rotations. If `step` is positive, the routine takes the top-most sibling window and makes it the last sibling window for each instance of `step`. If `step` is negative, the routine takes the last sibling window and makes it first. If `step` is zero, no rotations are performed. See the figures below.

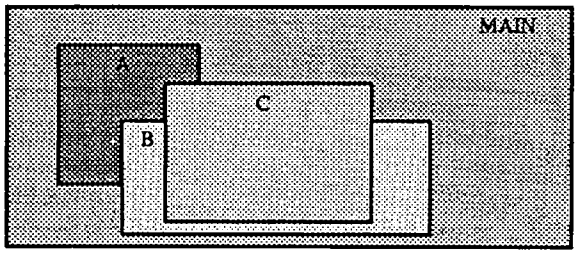

Figure 1: Screens `a`, `b`.and `c` are all siblings.Screen `main` is not a sibling.

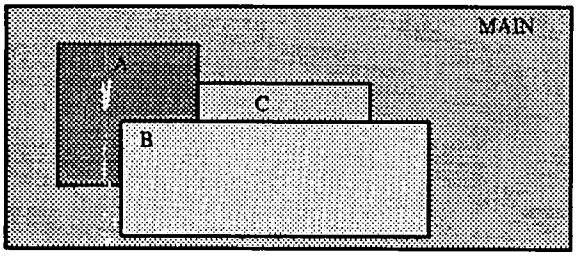

Figure 2: Executing `sm_wrotate (1)` rotates the top sibling to the bottom of the sibling stack. It rotates screen `c` behind the other two sibling windows, leaving screen `b` on top. Screen `main` is not affected.

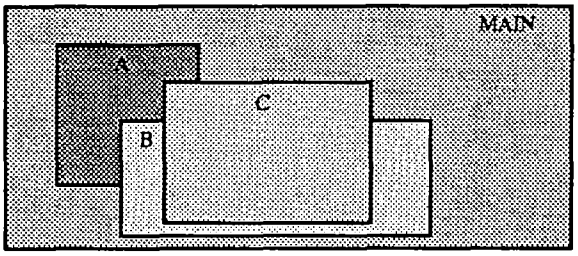

Figure 3: Executing `sm_wrotate (-1)` rotates the last sibling window to the top, putting screen `c` on top. The display is the same as Figure 1.

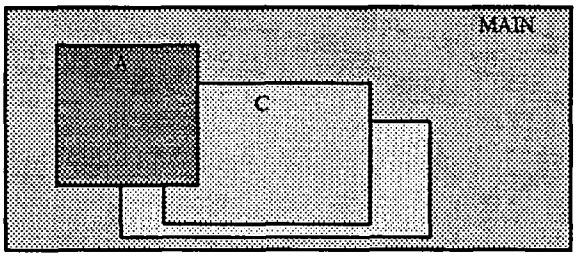

Figure 4: Executing `sm_wrotate (2)` rotates the first two sibling windows off the the top. First it rotates screen `c` to the back, then screen `b`, leaving screen `a` on top.

RETURNS

One less than the number of sibling windows on top of the window stack.
0 if there are no sibling windows

RELATED FUNCTIONS

```
call xsm_sibling(should_it_be);
```