ObjectStore®
PSE Pro™

PSE Pro for C++ Release Notes

Release 6.3

PROGRESS SOFTWARE | Real Time Division

PSE Pro for C++ Release Notes, Release 6.3, October 2005

September 2005

Contents

Preface 5

Release Notes 9

New and Changed Features 9

- New Platforms for PSE Pro for C++ 9
- Utility for Verifying Databases 10
- Environment Variable for Handling Exceptions 10
- Finding Invalid Pointers After Relocation 11
- Changed Signatures for self_check() Functions 11
- Changed Code for Collections Example 12

Known Problems 12

- HP-UX Possible Database Incompatibility 14

Compatibility of PSE Pro and ObjectStore 14

- API Additions in PSE Pro But Not in ObjectStore 14

Browsing and Searching the Documentation 16

- Using the HTML Documentation Set 16
- Using the PDF Documentation Set 16

Changes and Bug Fixes in Earlier Releases 17

- New Feature in 6.2 17
- New Features in 6.1.2 17
- Changes in Release 4.0 17
- Changes in Release 3.0.3 18
- Changes in Release 3.0.2 18
- New APIs in Release 3.0.2 19
- Bug Fixes in 6.2 20
- Bug Fixes in 6.1.2 20
- Bug Fixes in Release 4.0 21
- Bug Fixes in Release 3.0.3 21
- Bug Fixes in Release 3.0.0 23

Preface

Purpose

PSE Pro for C++ Release Notes provides information about new features, fixed problems, known problems, upgrading to Release 6.3, compatibility between PSE Pro and ObjectStore, and how to obtain support. For last-minute considerations, see the `readme` file in your PSE Pro installation directory.

Audience

This book is for programmers responsible for writing database applications that use PSE Pro for C++. It is assumed that you are experienced at developing C++ applications.

Scope

This book supports Release 6.3 of PSE Pro for C++.

Notation Conventions

This document uses the following conventions:

Convention	*Meaning*
`Courier`	`Courier font` indicates code, syntax, file names, API names, system output, and the like.
`Bold Courier`	**`Bold Courier font`** is used to emphasize particular code, such as user input.
`Italic Courier`	*`Italic Courier font`* indicates the name of an argument or variable for which you must supply a value.
Sans serif	Sans serif typeface indicates the names of user interface elements such as dialog boxes, buttons, and fields.
Italic serif	In text, *italic serif typeface* indicates the first use of an important term.
`[ ]`	Brackets enclose optional arguments.
`{ }*` or `{ }+`	When braces are followed by an asterisk (`*`), the items enclosed by the braces can be repeated `0` or more times; if followed by a plus sign (`+`), one or more times.
`{ a | b | c }`	Braces enclose two or more items. You can specify only one of the enclosed items. Vertical bars represent OR separators. For example, you can specify *a* or *b* or *c*.
`...`	Three consecutive periods can indicate either that material not relevant to the example has been omitted or that the previous item can be repeated.

Progress Software Real Time Division on the World Wide Web

The Progress Software Real Time Division Web site (www.progress.com/realtime) provides a variety of useful information about products, news and events, special programs, support, and training opportunities.

Technical Support

To obtain information about purchasing technical support, contact your local sales office listed at www.progress.com/realtime/techsupport/contact, or in North America call 1-781-280-4833. When you purchase technical support, the following services are available to you:

- You can send questions to realtime-support@progress.com. Remember to include your serial number in the subject of the electronic mail message.
- You can call the Technical Support organization to get help resolving problems. If you are in North America, call 1-781-280-4005. If you are outside North America, refer to the Technical Support Web site at www.progress.com/realtime/techsupport/contact.
- You can file a report or question with Technical Support by going to www.progress.com/realtime/techsupport/techsupport_direct.
- You can access the Technical Support Web site, which includes
  - A template for submitting a support request. This helps you provide the necessary details, which speeds response time.
  - Solution Knowledge Base that you can browse and query.
  - Online documentation for all products.
  - White papers and short articles about using Real Time Division products.
  - Sample code and examples.
  - The latest versions of products, service packs, and publicly available patches that you can download.
  - Access to a support matrix that lists platform configurations supported by this release.
  - Support policies.
  - Local phone numbers and hours when support personnel can be reached.

Education Services

To learn about standard course offerings and custom workshops, use the Real Time Division education services site (www.progress.com/realtime/services).

If you are in North America, you can call 1-800-477-6473 x4452 to register for classes. If you are outside North America, refer to the Technical Support Web site. For information on current course offerings or pricing, send e-mail to classes@progress.com.

Searchable Documents

In addition to the online documentation that is included with your software distribution, the full set of product documentation is available on the Technical Support Web site at www.progress.com/realtime/techsupport/documentation. The site provides documentation for the most recent release and the previous supported release. Service Pack README files are also included to provide historical context for specific issues. Be sure to check this site for new information or documentation clarifications posted between releases.

Your Comments

Real Time Division product development welcomes your comments about its documentation. Send any product feedback to `realtime-support@progress.com`. To expedite your documentation feedback, begin the subject with `Doc:`. For example:

```
Subject: Doc: Incorrect message on page 76 of reference manual
```

Release Notes

This document discusses the following topics:

New and Changed Features 9
Known Problems 12
Compatibility with Earlier PSE Pro for C++ Releases 14
Compatibility of PSE Pro and ObjectStore 14
Browsing and Searching the Documentation 16
Changes and Bug Fixes in Earlier Releases 17

New and Changed Features

PSE Pro Release 6.3 contains the following new and changed features:

- New Platforms for PSE Pro for C++
- Schema Generator Can Generate Schema Libraries
- Utility for Verifying Databases
- Environment Variable for Handling Exceptions
- Finding Invalid Pointers After Relocation
- Changed Signatures for self_check() Functions
- Changed Code for Collections Example

New Platforms for PSE Pro for C++

PSE Pro for C++ is available on the following new platforms:

- Linux 64
- Windows Visual Studio .NET 2003

On Windows platforms, only Visual Studio .NET 2003 is supported; Visual Studio .NET 2002 and Visual C++ 6.0 are no longer supported.

Schema Generator Can Generate Schema Libraries

You can now use the `pssg` utility to generate a schema library. For applications that run on Windows, a schema library is a dll. For applications that run on UNIX, a schema library is a shared library. You can use a schema library in the following ways.

- For verifying a database. See *PSE Pro for C++ API User Guide*, Checking Databases with the ospseverifydb Utility.
- In place of linking with the traditional schema object file in a PSE Pro application. When your program loads the schema library, PSE Pro initializes the typespecs for classes that are marked in the schema library.

For details about specifying the options that generate schema libraries, see *Building Applications with PSE Pro for C++*, Format of the Schema Generator Command Line.

Utility for Verifying Databases

To verify a PSE Pro database, execute the `ospseverifydb` utility. The command line format is as follows:

`ospseverifydb` *`db`* `[-schema_lib` *`schema_library`*`...]`

Replace *`db`* with the path of the PSE Pro database that you want to verify. If the database you want to verify contains only built-in types and no user-defined types, you need to specify only the database path.

If the database you want to verify contains any user-defined types, you must specify the `-schema_lib` option and replace *`schema_library`* with the path for one or more schema libraries. Specify the schema libraries that the `ospseverifydb` utility needs to verify the database. If you do not specify at least one schema library, the utility verifies C++ built-in types until it finds a user-defined type. At which point the utility throws `os_err_umarked_type` exception and terminates.

Information about the `ospseverifydb` utility is in *PSE Pro for C++ API User Guide*, Checking Databases with the ospseverifydb Utility. Information about generating schema libraries is in *Building Applications with PSE Pro for C++*, Running the Schema Generator.

Environment Variable for Handling Exceptions

Setting the `OS_PSE_DEF_EXCEP_ACTION` environment variable specifies an action that you want PSE Pro to perform when it is about to throw an exception.

On UNIX, setting this environment variable makes it easier to debug unhandled exceptions. The value you specify determines whether PSE Pro returns control to the debugger, dumps core, or exits with a particular return value.

By default, this variable is not set. When PSE Pro encounters an exception and this variable is not set, PSE Pro throws the exception to the program. If the program does not catch and handle the exception, the PSE Pro fault handler macros (`OS_PSE_ESTABLISH_FAULT_HANDLER` and `OS_PSE_END_FAULT_HANDLER`) print a message and then terminate the program.

You can specify the following values to set this variable:

Value	*Description*
`abort`	PSE Pro aborts the process. On UNIX, PSE Pro creates a core file and, if you are running in a debugger, returns control to the debugger.
`integer`	Specify an integer greater than or equal to 1. PSE Pro exits the program with the specified integer as the return value.
`kill`	PSE Pro action varies by platform: • Windows — `ExitProcess (0x006600);` • UNIX — `kill (getpid(), SIGKILL);`
Any other value	PSE Pro exits the program with a return value of `1`. This is the default value for setting this variable.

Finding Invalid Pointers After Relocation

The `obectstore::set_check_illegal_pointers()` function indicates whether you want PSE Pro to find any invalid pointers in a page after relocation. The function signature is

```
void set_check_illegal_pointers(os_boolean yes_no);
```

After you call this function with a true value, then after mapping a page into memory and performing relocation, PSE Pro checks the pointers in the relocated page to determine whether they are valid. PSE Pro displays a message that indicates any invalid pointers, and then continues processing. PSE Pro does not fix any invalid pointers.

The `OS_PSE_CHK_ILLEGAL_PTR` environment variable serves the same purpose as a call to the `set_check_illegal_pointers()` function.

The default value for the function call and for the environment variable is false.

Changed Signatures for self_check() Functions

The `os_database::self_check()` function takes a new argument. The function signature is now

```
void self_check(os_boolean b_skip_vtbls = 0);
```

Set the `b_skip_vtbls` argument to true when your program performs only database verification and the class definitions are not linked into the program.

The `objectstore::self_check()` function also takes the same new argument. The function signature is now

```
static void self_check(os_boolean b_skip_vtbls = 0);
```

For details about the `self_check()` functions, see *PSE Pro for C++ API User Guide*, Checking for Data Corruption.

Changed Code for Collections Example

In the PSE Pro collections example, the `dnary.hh` and `dnary.cc` files were modified to use the placement `new` operator to create a collection.

Known Problems

12994 pssg

For static builds, when linking `pssg`'s `osschm.obj` with the static PSE library, the linker will warn about `LNK4049`:

```
locally defined symbol “const os_pse::os_basic_typespec::‘vftable’” imported
```

The warning can be ignored.

13196 cpse

During `os_database::undo/redo()`, the fetch hook for the objects on undo/redo-pages is NOT being called.

13175 pssg

`pssg` crashes with STL `map<key..>` where `key` is enum.

When marking a template instantiation where the template actual argument is an enum value, `pssg` will crash. For example:

```
typedef const enum { K1=1; } KEY;
```

A workaround is to use an integer value instead, and instantiate the template with `"K1"`:

```
#define K1 1
typedef int KEY
```

13611 pssg

Crash when marking template `<int* p> class Foo`

When marking something like the following instantiation with a template actual argument that is a variable, `pssg` will crash accessing 0x0.

```
template <int *p> class Foo {}; int i; OS_MARK_SCHEMA_TYPE(Foo<&i>)
```

There is no workaround.

13700 pssg

Not patching long type names: “`void*`” and “`short int`” (STL map)

When supplying “`long`” class names explicitely in the `.scm` file (compare with `stl/stl42/readme.txt`), pointers need to be specified as “`void *`” instead of “`void*`” (there must be a space between void and *), otherwise the linker will

complain about a missing symbol `x::the_instance` where "`x`" is the "`long`" class name being marked.

Similarly, when instantiation templates, use "`short`" instead of "`short int`".

13721 cpse

Invalid `os_ts<T>` members if `DllEntryPoint()` present in DLL

When building a DLL that contains marked types and therefore `os_ts<T>` instances, do not add a function `DllEntryPoint()` with the signature

```
BOOL WINAPI DllEntryPoint(HINSTANCE, DWORD, LPVOID)
```

This prevents the static constructors of the `os_ts<T>` instances from being run in the `osschm.obj` that `pssg` creates. A typical error is that `typespec` instances return invalid values. For example:

```
os_typespec *ts = os_ts<Foo>::get(); int sz = ts->get_size(); //
returns 2328371 instead of 8
```

13930 pssg

`pssg` crashes when using `#import` statements in `.scm` file

`pssg` crashes if the `.scm` file contains a `#import` directive. To work around this problem, conditionalize the directive using `_ODI_PSSG_`, for example:

```
#ifndef _ODI_PSSG_ #import "mylib.tlb"  no_implementation #endif
```

Compatibility with Earlier PSE Pro for C++ Releases

PSE Pro for C++ Release 6.3 is compatible with applications and databases built with PSE Pro for C++ Release 6.*x*, 5.0 and 4.0. You can have Release 6.3 and Release 6.x, 5.1, and 5.0 installed on the same machine. Databases created with any release are compatible.

For PSE Pro for C++ Release 5.1 and Release 5.0, the PSE Pro DLLs have been renamed to have the `o5*` prefix to distinguish them from Release 4.0 DLLs.

On the HP-UX platform the schema format has been modified to support virtual base classes, so customers need to re-generate schemas, and then recompile and re-link the schema.

HP-UX Possible Database Incompatibility

A bug in version 3.45 of HP's aCC compiler can cause a database incompatibility between PSE Pro 6.3 and previous releases of PSE Pro on HP-UX. For some applications that have schemas that contain virtual base classes, the database schema generated by previous releases of PSE Pro is incompatible with the database schema generated by PSE Pro 6.3.

A detailed explanation of the bug and its fix is on the HP Web site at `http://h21007.www2.hp.com/dspp/tech/tech_TechDocumentDetailPage_IDX/1,1701,7866,00.html#faq-compat)`.

To determine whether your application will have this incompatibility, use PSE Pro 6.3's schema generator to regenerate your application's schema. Then compare the object schema file (the `.cpp` file) generated by 6.3 with the object schema file generated by your previous version of PSE Pro.

If you find that the two schemas are incompatible, you must evolve your database to be able to use it with PSE Pro 6.3.

Compatibility of PSE Pro and ObjectStore

The PSE Pro API is a subset of the ObjectStore API. In general, you should have no trouble upgrading from PSE Pro to ObjectStore. However, the PSE Pro API differs from that of ObjectStore in a few ways.

API Additions in PSE Pro But Not in ObjectStore

PSE Pro performs some tasks differently from ObjectStore. Consequently, the PSE Pro API includes additions to its interface that are not found in the ObjectStore API:

- `os_database::save()`

  ObjectStore requires the use of transactions. With ObjectStore, call `os_transaction::commit()` to save the state of the database.

- os_database::save_as() and os_database::copy()

  With ObjectStore, use the oscopy utility.

- os_database::self_check() and objectstore::self_check()
- Undo/redo facility
  - os_database::set_undo_marker()
  - objectstore::set_maximum_undo_operations()
  - os_database::undo()
  - os_database::redo()
  - os_database::is_altered()
  - os_database::is_undoable()
  - os_database::is_redoable()
- Specification of istorage, stream_name, and reserve_mb_adr_space parameters when you create or open a database.
- objectstore::disable_logging()
- Schema evolution facility
  - os_database::rename_type()
  - os_database::record_new_object_location()
  - os_database::fix_pointers()
- os_database::needs_recovery() and os_database::recover()
- os_database::dump_contents()
- os_database::is_modified()
- os_get_page_stats()

Other differences between PSE Pro and ObjectStore include the following:

- With ObjectStore, transactions are required. With PSE Pro, transactions are optional.
- ObjectStore supports nested transactions. PSE Pro does not.
- ObjectStore uses the TIX_EXCEPTION handler. PSE Pro uses native exceptions.
- ObjectStore environment variables and macros start with OS_. PSE Pro environment variables and macros start with OS_PSE_.
- ObjectStore's schema generator, ossg, creates

  T::get_os_typespec()

  PSE Pro's schema generator, pssg, creates

  os_ts<T>::get()
- In ObjectStore, os_database::is_open() is not a static function. In PSE Pro, it is static.

Browsing and Searching the Documentation

The documentation for PSE Pro is available in HTML and PDF formats. Each format is described in the following topics:

- Using the HTML Documentation Set
- Using the PDF Documentation Set

The HTML version of the documentation set contains a full text search feature.

Using the HTML Documentation Set

To open the PSE Pro documentation set on Windows platforms, select the Start | Programs | ObjectStore PSE Pro for C++ | PSE Pro Bookshelf menu item.

To open the PSE Pro documentation set on UNIX platforms, open the `doc/index.htm` file in a browser. The full pathname for this file is typically `/opt/ODI/PSEProC/doc/index.htm`.

To use the HTML version of the PSE Pro for C++ documentation set, you need to have JavaScript enabled.

The HTML version of the PSE Pro documentation set is tested on Internet Explorer 6.0.x, Netscape 7.0.x, and Mozilla Firefox 1.0.x. The documentation set *cannot* be displayed properly using Netscape 6.0.

Using the PDF Documentation Set

If you want to view the documentation set in PDF format or if you want print a hard copy of the documentation, open the PDF versions of the books that are located in the `doc/pdf` directory. You can also select PDF versions of the documentation by opening the `doc/index.htm` file in a browser. The API reference information is generated with the javadoc tool; therefore, PDF version of that information is not available.

To view PDF files in Acrobat Reader, you must use Acrobat Reader 5.0 or a subsequent release.

Changes and Bug Fixes in Earlier Releases

This section provides information about changes and bug fixes in earlier PSE Pro releases.

New Feature in 6.2

PSE Pro now uses the standard specification for streams in place of the classic specification. Where your code uses the following:

```
#include <iostream.h>
#include <fstream.h>
```

You must change it to this:

```
#include <iostream>
#include <fstream>
```

After you make these changes, you must re-compile and re-link your application.

New Features in 6.1.2

PSE Pro for C++ is now supported on the Solaris platform.

The PSE Pro for C++ schema generator (`pssg`) is fully supported on on all platforms.

Virtual base classes are now supported in PSE Pro for C++.

Changes in Release 4.0

os_database::is_altered()

```
os_boolean is_altered();
```

This function deteremines whether or not a database has been modified after an undo marker was set; it returns non-zero (true) if the database was modified after the last call to this function or after an `os_database::set_undo_marker()` is called. You can call this function to avoid setting unnecessary undo markers when a database has not been modified.

Support for Variable Page Size Settings

The new `OS_PSE_PAGE_SIZE` environment variable enables you to set the PSE Pro database page size. Setting the PSE Pro database page size (or *soft page size*) to a higher value than the default of 4K generally increases performance and increases the maximum allowed address range that could be mapped.

To use this feature, set the environment variable `OS_PSE_PAGE_SIZE` to the value in kilobytes of the page size. For example, to set a page size of 32K on a UNIX platform, use

```
# setenv OS_PSE_PAGE_SIZE 32
```

For complete information, see `OS_PSE_PAGE_SIZE` in the *PSE Pro for C++ API User Guide*.

Improved Paging Scheme

The PSE Pro internal paging scheme has been improved to use a hash function. This enables faster access to database pages and provides increased scalability for large databases.

New Rogue Wave STL Example

A new example application shows how to use Rogue Wave's Standard Template Library (STL) persistently with PSE Pro. See `readme.txt` in the `examples/rwstl` directory for information on how to use this example.

Support for ObjectStore Inspector

With the PSE Pro version of the ObjectStore Inspector you can browse, edit, and report on data in a PSE Pro database. Inspector comes with a complete online help system that describes how to use it. In addition, see *Using PSE Pro for C++ Inspector* for a brief introduction to Inspector.

Changes in Release 3.0.3

Another approach to map pages using `mapaside` has been implemented for this Service Pack. Mapaside relocates a page at a `mapaside` buffer and then copies it over to the page being mapped. This considerably reduces the block for which threads are frozen (only for the duration when the page is mapped and the mapaside buffer copied). But `mapaside` has overheads over in-place relocation, that of copying each page relocated from the mapaside buffer to the page mapped.

`mapaside` can be enabled/disabled using the API or by setting an environment variable.

- At runtime using the API `objectstore::set_mapaside(os_boolean yes_no)`. Use `objectstore::set_mapaside(true)` to enable `mapaside`. The default value for the API call is true.
- By setting the environment variable `OS_PSE_DISABLE_MAPASIDE`. `mapaside` can be disabled by setting `OS_PSE_DISABLE_MAPASIDE` to `1`.

PSE Pro, by default, uses the `mapaside` scheme to map pages.

Changes in Release 3.0.2

- Collection schema and DLL are incompatible with previous releases. They are now equal to ObjectStore/C++ 6.0 Service Pack 1. That is, modules using the collection API have to be recompiled and relinked against PSE 3.0 Service Pack 2. Existing databases containing collection objects cannot be used with PSE 3.0 Service Pack 2. Databases have to be recreated.
- Collection memory leaks have been resolved (Amber 14602).
- An application using ObjectStore/C++ *and* PSE C++ can now use the Collection API as well, since the `os_pse` namespace has been added to PSE C++ Collections (Amber 14127). Refer to the PSE Pro *ostore* example, and note the following order of include files:

a Include files for ObjectStore

b Include files for ObjectStore collections

c Include files for PSE.

d Include files for PSE collections

The ObjectStore/PSE Pro C++ collection headers reuse types and macros and so this order must be strictly maintained so that the corresponding files pick up the correct definitions for the types and macros.

New APIs in Release 3.0.2

New and modified APIs in Release 3.0 Service Pack 2 are described briefly here.

os_database::get_n_roots()
os_database::get_all_roots()

```
os_int32 get_n_roots() const;
void get_all_roots(
     os_int32 max_roots,
     os_database_root_p*roots,
     os_int32& n_roots) const;
```

Provides access to all the roots in the specified database The `os_database_root_p*` is an array of pointers to roots. This array must be allocated by the user. The function `os_database::get_n_roots()` can be used to determine the size of an array to allocate. `max_to_return` is specified by the user and is the maximum number of elements the array is to have.

objectstore::
set_thread_locking_during_relocation(os_boolean yes_no)
OS_PSE_DISABLE_THREAD_LOCKING=1

In circumstances where multiple threads are used and where these threads might operate on the same database, it is possible that, after a thread faults on a page and during the process of mapping and relocating that page, another thread might access the same page. in such a case, the second thread is not reading consistent data. To prevent this from happening, when a page is being relocated, all other threads are suspended(frozen) before relocation starts and resumed after relocation is complete. This may cause deadlock situations. You can skip freezing of threads to avoid deadlock by using the `objectstore::set_thread_locking_during_relocation()` api with a false parameter. The user is then responsible for providing for proper thread synchronization. You can skip thread locking without problems in applications where different threads operate on different databases.

The default value for the API call is false. Thread locking can also be set using the environment variable `OS_PSE_DISABLE_THREAD_LOCKING`. Thread locking is skipped if `OS_PSE_DISABLE_THREAD_LOCKING` is set to 1.

objectstore::
set_check_illegal_pointers(os_boolean yes_no)

OS_PSE_CHK_ILLEGAL_PTR=1

This API sets the boolean value to checking for validation of invalid pointers in the database. If `set_check_illegal_pointers()` is called with a true value, then, on relocation in a page after a page is mapped in memory, validation of all the pointers in that page is carried out.

The default value for the API call is false. The environment variable `OS_PSE_CHK_ILLEGAL_PTR` can also be used to set checking for validation of invalid pointers in the database.

Change to os_database::lookup()

```
static os_database *lookup(
  const char *pathname,
  os_int32 create_mode = 0
);
```

Returns a pointer to the database with the specified `pathname` but does not open it. If it is not found, an `err_database_not_found` exception is signaled. `create_mode` is a Boolean value; if its value is nonzero and no database named `pathname` exists, it creates the database.

Bug Fixes in 6.2

18696 cpse

In the previous release, PSE Pro threw an exception when an application aborted a transaction that used large objects. PSE Pro no longer throws this exception.

18485 cpse

The `objectstore::initialize()` function no longer sets the locale. It is set only when handling filenames and there after reset back to the original.

Bug Fixes in 6.1.2

13165 pssg

Previously, `pssg` created wrong relocation info for vtbl in classes w/ virtual bases. The new `pssg` supports virtual base classes.

17004 cpse

The `os_databse::is_altered()` method did not return `true` on data modification. The `os_database::is_altered()` method returned `true` on creation or deletion of objects but not when an object was modified. This has been corrected.

17036 cpse

Check for adjusted size of allocation incorrect in `_ODI_change_typespec()`

17268 cpse

In UNIX versions, an error message was printed for missing vtbl symbol even if `OS_PSE_DEBUG_VTBL` was not set.

17326 cpse

In UNIX versions, the cache file was hardcoded to the `/tmp` directory.

Bug Fixes in Release 4.0

15814 Usage of the `OS_PSE_CHK_ILLEGAL_PTR` environment variable when `mapaside` is enabled caused exception `os_err_database_transient`

15829 Under certain circumstances the thread local storage (TLS) entries for exited threads were not properly deleted, which led to deadlocks (case number 90962).

16094 Calls to `os_transaction::abort()` unmapped system pages, which caused a recursive fault.

15711 Program group items didn't work if PSE Pro was installed in the `\Program Files` directory.

14868 Failure when creating a huge database threw an `os_err_internal` error instead of an `os_err_out_of_virtual_memory` error.

Bug Fixes in Release 3.0.3

14760 ***15614*** Multiple threads accessing the same database could cause deadlocks.

15755 Calls to os_transaction::abort() could fail to relocate pages when a transaction was aborted.

Bug Fixes in Release 3.0.2

14025 test/ostore: macro redefinition (txn and exception stuff)

14033 add resource information to DLLs (bad resource for `o3mfcu.dll`)

14132 VC++ debug C-runtime reports 20B leak on the global `os_transaction` object

14594 internal thread-local-storage leaks thread-handle

14761 `return_all_pages(): os_err_internal - Assertion` "n < n_bits" utils\bitvec.cc

14936 undefined `os_ts<bool>::get()`, needs to be added to `include/os_pse/api/os_tspec`.

14951 `os_database::lookup()` has wrong read_only arg

15055 examples/ostore doesn't work with OStore R6

15057 PCR app-schema-dll support for PSE C++ MOP

14035 `os_Dictionary::pick()` returns 0

14127 PSE & Collection app not supported (chlist.hh(579): cannot convert to `os_coll_ptr`)

14128 btree based `os_Dictionary` crashes w/ "Calculated filler size .."

14129 dtest/`dtest -t_pchar` fails in `d_chpt_dups_ordered`, test_reorg

14466 `_Long_hash` and `_Long_rank` not exported

14517 PSE pro link error - `os_rel_many::no_value_descriptor()` not implemented

14602 Case Number 76406 mem leaks in PSE collections

14835 Case Number 81632 : Unresolved external `os_rel_protect_being_deleted`

14400 3.0 pssg on VC50 crashes with at instruction at "0x00000020", memory could not be read

13552 SDI/MDI apps don't remove db/log in \temp

14802 crash in OnOpenDocument->WideCharToMultiByte in OSMFC, Unicode

15323 Log file being deleted by `os_database::open()` attempting to open a corrupted database. Previous workaround of copying the log file just before attempting to open the database is no longer needed.

Bug Fixes in Release 3.0.1

14197 #SE068059_O# relat.hh in PSE Pro C++ 3.0

Bug Fixes in Release 3.0.0

14139 custom build rules for PSSG not found, examples/`schema_evol`, VC60

12572 40B mem leak in `util.cc: DllMain()`, PROCESS_ATTACH

14122 crash w/ `os_Dictionnary<char*, B*>::insert()`

14138 VC++ 6.0 recognized: LNK4064: conflicting subsystem

14130 linking STL example on VC++ 6.0: corrupted file `osschm.obj`

O

objectstore::set_check_illegal_pointers() 11
`OS_PSE_CHK_ILLEGAL_PTR` environment variable 11