



Persistent State Service Reference

Version 6.2, December 2004

IONA, IONA Technologies, the IONA logo, Orbix, Orbix/E, Orbacus, Artix, Orchestrator, Mobile Orchestrator, Enterprise Integrator, Adaptive Runtime Technology, Transparent Enterprise Deployment, and Total Business Integration are trademarks or registered trademarks of IONA Technologies PLC and/or its subsidiaries.

Java and J2EE are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

CORBA is a trademark or registered trademark of the Object Management Group, Inc. in the United States and other countries. All other trademarks that appear herein are the property of their respective owners.

While the information in this publication is believed to be accurate, IONA Technologies PLC makes no warranty of any kind to this material including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IONA Technologies PLC shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

COPYRIGHT NOTICE

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of IONA Technologies PLC. No third party intellectual property right liability is assumed with respect to the use of the information contained herein. IONA Technologies PLC assumes no responsibility for errors or omissions contained in this book. This publication and features described herein are subject to change without notice.

Copyright © 2001–2004 IONA Technologies PLC. All rights reserved.

All products or services mentioned in this manual are covered by the trademarks, service marks, or product names as designated by the companies who market those products.

Updated: 07-Dec-2004

Contents

CosPersistentState Overview	1
CosPersistentState::AccessMode Type	2
CosPersistentState::ForUpdate Enumeration	2
CosPersistentState::IsolationLevel Type	3
CosPersistentState::NotFound Exception	4
CosPersistentState::Parameter Structure	4
CosPersistentState::ParameterList Sequence	5
CosPersistentState::Pid Type	5
CosPersistentState::ShortPid Type	5
CosPersistentState::TransactionalSessionList Sequence	5
CosPersistentState::TypeId Type	6
CosPersistentState::YieldRef Enumeration	6
CosPersistentState::CatalogBase Interface	9
CatalogBase::access_mode Attribute	9
CatalogBase::close()	10
CatalogBase::find_by_pid()	10
CatalogBase::find_storage_home()	10
CatalogBase::flush()	11
CatalogBase::free_all()	11
CatalogBase::refresh()	12
CosPersistentState::Connector Interface	13
Connector::create_basic_session()	14
Connector::create_transactional_session()	16
Connector::current_session()	18
Connector::get_pid()	18
Connector::get_short_pid()	18
Connector::implementation_id Attribute	19
Connector::register_session_factory()	19
Connector::register_session_pool_factory()	19
Connector::register_storage_home_factory()	20
Connector::register_storage_object_factory()	20
Connector::sessions()	20

Table of Contents

CosPersistentState_Factory Template	23
CosPersistentState::EndOfAssociationCallback Interface	25
CosPersistentState::Session Interface	27
CosPersistentState::StorageHomeBase Interface	29
StorageHomeBase::find_by_short_pid()	29
StorageHomeBase::get_catalog()	30
CosPersistentState::StorageHomeFactory Native Type	31
CosPersistentState::StorageObject Interface	33
StorageObject::destroy_object()	33
StorageObject::get_pid()	33
StorageObject::get_short_pid()	34
StorageObject::get_storage_home()	34
StorageObject::object_exists()	34
CosPersistentState::StorageObjectBase Native Type	35
CosPersistentState::StorageObjectFactory Native Type	37
CosPersistentState::StorageObjectRef Class	39
StorageObjectRef::_catalog()	40
StorageObjectRef::destroy_object()	40
StorageObjectRef::get_pid()	41
StorageObjectRef::get_short_pid()	41
StorageObjectRef::get_storage_home()	41
StorageObjectRef::_impl_data()	41
StorageObjectRef::is_null()	41
StorageObjectRef::operator=()	41
StorageObjectRef::operator->()	42
StorageObjectRef::release()	42
StorageObjectRef::same_ref()	42
StorageObjectRef::_static_type()	42

StorageObjectRef::StorageObjectRef() Constructors	42
StorageObjectRef::_target_type	43
StorageObjectRef::_target()	43
CosPersistentState::TransactionalSession Interface	45
TransactionalSession::AssociationStatus Type	46
TransactionalSession::default_isolation_level Attribute	46
TransactionalSession::end()	46
TransactionalSession::get_association_status()	47
TransactionalSession::start()	47
TransactionalSession::suspend()	48
TransactionalSession::transaction()	49
IT_PSS Overview	51
IT_PSS::CatalogBase Interface	53
CatalogBase::it_create_statement()	54
CatalogBase::it_create_statement_with_type_and_concurrency()	54
CatalogBase::it_discard_all()	54
CatalogBase::it_discard_flush_list()	55
CatalogBase::it_prepare_statement()	55
CatalogBase::it_prepare_statement_with_type_and_concurrency()	55
IT_PSS::Connector Interface	57
Connector::it_create_session_manager()	57
IT_PSS::DynamicReplica Interface	59
IT_PSS:Master Interface	61
IT_PSS::PreparedStatement Interface	63
PreparedStatement::clear_parameters()	63
PreparedStatement::define_parameter()	64
PreparedStatement::execute_prepared()	64
PreparedStatement::execute_prepared_query()	64
PreparedStatement::execute_prepared_update()	64

Table of Contents

IT_PSS:Replica Interface	67
IT_PSS::Replica::set_master	67
IT_PSS::Replica::last_successful_refresh	68
IT_PSS:Replica:refresh	68
IT_PSS::ResultSet Interface	69
ResultSet::absolute()	72
ResultSet::after_last()	72
ResultSet::before_first()	73
ResultSet::cancel_row_updates()	73
ResultSet::close()	73
ResultSet::Concurrency Type	73
ResultSet::delete_row()	74
ResultSet::FetchDirection Type	74
ResultSet::find_state_member()	74
ResultSet::first()	75
ResultSet::get()	75
ResultSet::get_by_name()	75
ResultSet::get_concurrency()	76
ResultSet::get_fetch_direction()	76
ResultSet::get_fetch_size()	76
ResultSet::get_row()	76
ResultSet::get_statement()	77
ResultSet::get_type()	77
ResultSet::insert_row()	77
ResultSet::is_after_last()	77
ResultSet::is_before_first()	77
ResultSet::is_first()	78
ResultSet::is_last()	78
ResultSet::last()	78
ResultSet::move_to_current_row()	78
ResultSet::move_to_insert_row()	79
ResultSet::next()	79
ResultSet::previous()	79
ResultSet::refresh_row()	79
ResultSet::relative()	80
ResultSet::row_deleted()	80
ResultSet::row_inserted()	80

ResultSet::row_updated()	80
ResultSet::set()	81
ResultSet::set_by_name()	81
ResultSet::set_fetch_direction()	81
ResultSet::set_fetch_size()	82
ResultSet::Type	82
ResultSet::update_row()	83
IT_PSS::Session Interface	85
IT_PSS::SessionManager Interface	87
SessionManager::get_shared_read_only_session_nc()	87
SessionManager::block_readers_until_idle()	87
IT_PSS::Statement Interface	89
Statement::close()	90
Statement::execute()	90
Statement::execute_query()	90
Statement::execute_update()	91
Statement::get_catalog()	91
Statement::get_fetch_direction()	91
Statement::get_fetch_size()	91
Statement::get_result_set()	92
Statement::get_result_set_concurrency()	92
Statement::get_result_set_type()	92
Statement::set_fetch_direction()	92
Statement::set_fetch_size()	93
IT_PSS_StorageHomeFactory Template	95
IT_PSS_StorageHomeFactory::_add_ref()	95
IT_PSS_StorageHomeFactory::_create()	96
IT_PSS_StorageHomeFactory::IT_PSS_StorageHomeFactory()	96
IT_PSS_StorageHomeFactory::_remove_ref()	96
IT_PSS::StorageObject Interface	97
StorageObject::it_lock()	97

Table of Contents

IT_PSS_StorageObjectFactory Template	99
IT_PSS_StorageObjectFactory::_add_ref()	99
IT_PSS_StorageObjectFactory::create()	99
IT_PSS_StorageObjectFactory::IT_PSS_StorageObjectFactory()	100
IT_PSS_StorageObjectFactory::_remove_ref()	100
IT_PSS::TransactionalSession Interface	101
IT_PSS::TransactionalSession::get_master	101
IT_PSS::TransactionalSession:is_replica	101
IT_PSS::TransactionalSession:get_replica	102
IT_PSS::TransactionalSession2 Interface	103
IT_PSS::TransactionalSession2::refresh_master	103
IT_PSS::TxSessionAssociation Class	105
TxSessionAssociation::end()	106
TxSessionAssociation::get_session_nc()	106
TxSessionAssociation::get_tx_coordinator_nc()	106
TxSessionAssociation::TxSessionAssociation() Constructors	107
TxSessionAssociation::~~TxSessionAssociation() Destructor	108
TxSessionAssociation::suspend()	108
The IT_PSS_DB Module Overview	109
IT_PSS_DB::Env Interface	111
Env::checkpoint()	111
Env::name Attribute	111
Env::post_backup()	112
Env::pre_backup()	112
Index	113

CosPersistentState Overview

The persistent state service (PSS) is a CORBA-friendly object-oriented database. PSS storage objects can hold any kind of IDL type. The Orbix implementation of PSS is organized into three modules and an object factory class:

- [“CosPersistentState Overview”](#)

The `CosPersistentState` module is the standard OMG service for persistent objects.

- [“IT_PSS Overview”](#)

The `IT_PSS` module provides various proprietary useful features such as queries.

- [The IT_PSS_DB Module Overview](#)

The Orbix implementation of PSS is targeted at relational and relational-like database back-ends. It is not restricted to any particular database system.

The `CosPersistentState` module's features are listed in [Table 1](#):

Table 1: *The CosPersistentState Module*

Common Data Types	Interfaces
AccessMode Type	CatalogBase
ForUpdate Enumeration	Connector
IsolationLevel Type	EndOfAssociationCallback
NotFound Exception	Session
Parameter Structure	StorageHomeBase
ParameterList Sequence	TransactionalSession
Pid Type	
ShortPid Type	
TransactionalSessionList Sequence	
TypeId Type	
YieldRef Enumeration	
	Native Types and Helper Classes
	CosPersistentState_Factory
	StorageHomeFactory
	StorageObjectBase
	StorageObjectFactory
	StorageObjectRef

The rest of this chapter describes the common data types for the module.

CosPersistentState::AccessMode Type

```
// PSDL Code
typedef short AccessMode;
```

The mode of access for a storage object. Valid values include:

```
READ_ONLY
READ_WRITE
```

The `AccessMode` `READ_WRITE` is higher than `READ_ONLY`.

CosPersistentState::ForUpdate Enumeration

```
// PSDL Code
enum ForUpdate { FOR_UPDATE };
```

Used in the language mapping to define an overloaded accessor method that can update the state member.

Examples

For example, a state member whose type is an abstract storagetype is mapped to a read-only accessor, a read-write (update) accessor, and a modifier:

```
// PSDL
abstract storagetype A {};
abstract storagetype B {
    state A embedded;
};
```

This PSDL code maps to:

```
// C++
class B : public virtual StorageObject {
public:
    virtual const A& embedded() const = 0;
    virtual A& embedded(CosPersistentState::ForUpdate) = 0;
    virtual void embedded(const A&) = 0; // copies
};
```

CosPersistentState::IsolationLevel Type

```
// PSDL Code
typedef short IsolationLevel;
const IsolationLevel READ_UNCOMMITTED = 0;
const IsolationLevel READ_COMMITTED = 1;
const IsolationLevel REPEATABLE_READ = 2;
const IsolationLevel SERIALIZABLE = 3;
```

When data is accessed through a transactional session actively associated with a resource, undesirable phenomena such as dirty reads or non-repeatable reads may occur. An isolation level controls user access to these kinds of phenomenon during a transactional session.

Valid `IsolationLevel` values include the following:

<code>READ_UNCOMMITTED</code>	When a resource has this isolation level, its user may experience the dirty reads and the non-repeatable reads phenomena.
<code>READ_COMMITTED</code>	When a resource has this isolation level, its user may experience the non-repeatable reads phenomenon, but not the dirty reads phenomenon.

SERIALIZABLE	When a resource has this isolation level, its user is protected from both the dirty reads and the non-repeatable reads phenomena
REPEATABLE_READ	This isolation level is reserved for future use.

A dirty read occurs when a resource is used to read the uncommitted state of a storage object. For example, suppose a storage object is updated using resource 1. The updated storage object's state is read using resource 2 before resource 1 is committed. If resource 1 is rolled back, the data read with resource 2 is considered never to have existed.

A non-repeatable read occurs when a resource is used to read the same data twice but different data is returned by each read. For example, suppose resource 1 is used to read the state of a storage object. Resource 2 is used to update the state of this storage object and resource 2 is committed. If resource 1 is used to reread the storage object's state, different data is returned.

See Also [CosPersistentState::TransactionalSession](#)

CosPersistentState::NotFound Exception

```
// PSDL Code
exception NotFound {};
```

An exception that indicates that a storage object or registry connector cannot be found.

CosPersistentState::Parameter Structure

```
// PSDL Code
struct Parameter {
    string name;
    any val;
};
```

A parameter in a list of parameters when creating a session.

Parameters

name The parameter's name.
val The value in the parameter.

See Also

[CosPersistentState::ParameterList](#)
[CosPersistentState::Connector::create_basic_session\(\)](#)
[CosPersistentState::Connector::create_transactional_session\(\)](#)

CosPersistentState::ParameterList Sequence

```
// PSDL Code  
typedef sequence<Parameter> ParameterList;
```

A sequence of [Parameter](#) structures.

See Also

[CosPersistentState::Parameter](#)

CosPersistentState::Pid Type

```
// PSDL Code  
typedef CORBA::OctetSeq Pid;
```

A global persistent object identifier that storage objects use. The scope of the `pid` is all storage objects that can be accessed through the same catalog.

See Also

[CosPersistentState::ShortPid](#)

CosPersistentState::ShortPid Type

```
// PSDL Code  
typedef CORBA::OctetSeq ShortPid;
```

A storage object identifier that is unique within a storage home family.

See Also

[CosPersistentState::Pid](#)

CosPersistentState::TransactionalSessionList Sequence

```
// PSDL Code
```

```
typedef sequence<TransactionalSession> TransactionalSessionList;
```

A list of transactional sessions.

See Also

[CosPersistentState::TransactionalSession](#)
[CosPersistentState::Connector::sessions\(\)](#)

CosPersistentState::TypeId Type

```
// PSDL Code
typedef string TypeId;
```

A string that identifies a PSDL type. The format of a PSDL type id is the same as the IDL format of repository ids, except that the prefix is PSDL, not IDL.

See Also

[CORBA::RepositoryId](#)
[CosPersistentState::Connector](#)

CosPersistentState::YieldRef Enumeration

```
// PSDL Code
enum YieldRef { YIELD_REF };
```

Used in the language mapping to define overloaded methods that yield incarnations and references as parameters.

Examples

For example, a state member whose type is a reference to an abstract storagetype is mapped to two accessors and two modifier methods:

```
// PSDL
abstract storagetype Bank;
abstract storagetype Account {
    state ref<Bank> my_bank;
};
```

The mapping shows that one of the accessor methods takes no parameter and returns a storage object incarnation, and the other takes a `YieldRef` parameter and returns a reference:

```
// C++
class Account : public virtual StorageObject {
public:
    virtual Bank* my_bank() const= 0;
    virtual const BankRef* my_bank(
```

```
        CosPersistentState::Yield-Ref yr
    ) const = 0;

    virtual void my_bank(Bank* b) = 0;
    virtual void my_bank(const BankRef* b) = 0;
};
```


CosPersistentState::CatalogBase Interface

The CatalogBase interface is the base interface for the implementation of a local catalog object.

```
// PSDL in module CosPersistentState
local interface CatalogBase {

    readonly attribute AccessMode access\_mode;

    StorageHomeBase find\_storage\_home(
        in string storage_home_type_id
    )
        raises (NotFound);

    StorageObjectBase find\_by\_pid(
        in Pid the_pid
    )
        raises (NotFound);

    void flush();
    void refresh();
    void free\_all();
    void close();
};
```

CatalogBase::access_mode Attribute

```
// PSDL code
readonly attribute AccessMode access_mode;
```

Returns the access mode of this catalog. When the access mode is `READ_ONLY`, the storage object incarnations obtained through storage home instances provided by this catalog are read-only.

CatalogBase::close()

```
// PSDL code
void close();
```

Terminates the catalog. If the catalog is associated with one or more transactions when `close()` is called, these transactions are marked as roll-back only. When closed, the catalog is also flushed for a non-transactional session.

CatalogBase::find_by_pid()

```
// PSDL code
StorageObjectBase find_by_pid(
    in Pid the_pid
)
    raises (NotFound);
```

Attempts to locate a storage object and returns an incarnation of it.

Parameters

`the_pid` The operation uses the given [pid](#) to find the storage object in the storage homes provided by the target catalog.

Exceptions

`NotFound` The operation cannot find a storage object with this `pid`.
 exception

CatalogBase::find_storage_home()

```
// PSDL code
StorageHomeBase find_storage_home(
    in string storage_home_type_id
)
    raises (NotFound);
```

Returns a storage home instance.

Parameters

`storage_home_type_id` The operation looks up a PSDL-defined storage home with this Id in the catalog's default data-store.

The format of this parameter is mostly implementation-defined. In the case of type-specific catalogs (declared in PSDL), the provided declarations define valid values for this parameter.

The operation can also interpret Ids that have the form of a PSDL type Id. For example:

```
PSDL:com/acme/PersonStoreImpl:1.0
```

Exceptions

`NotFound` The operation cannot find a storage home that matches the given storage home Id.

CatalogBase::flush()

```
// PSDL code
void flush();
```

Writes to disk any cached modifications of storage object incarnations managed by this catalog. PSS can cache some *dirty* data, thus, when an application creates a new storage object or updates a storage object, the modification is not written directly to disk.

CatalogBase::free_all()

```
// PSDL code
void free_all();
```

Instructs the catalog implementation to set the reference count of all its PSDL storage object instances to 0.

CatalogBase::refresh()

```
// PSDL code  
void refresh();
```

Refreshes any cached storage object incarnations accessed (read) by this catalog. In addition to caching write data, PSS can cache data read from datastores.

Note: This operation can invalidate any direct reference to a storage object incarnation's data member. Most applications do not use `refresh()`, so calling it is unusual.

CosPersistentState::Connector Interface

A connector is a local object that represents a given PSS implementation. Sessions are created by connectors. You obtain a connector of a given ORB by calling `CORBA::ORB::resolve_initial_references()` with the `ObjectId` of PSS.

```
// PSDL code in module CosPersistentState
local interface Connector {

    readonly attribute string implementation\_id;

    Pid get\_pid(
        in StorageObjectBase obj
    );

    ShortPid get\_short\_pid(
        in StorageObjectBase obj
    );

    Session create\_basic\_session(
        in AccessMode access_mode,
        in TypeId catalog_type_name,
        in ParameterList additional_parameters
    );

    TransactionalSession create\_transactional\_session(
        in AccessMode access_mode,
        in IsolationLevel default_isolation_level,
        in EndOfAssociationCallback callback,
        in TypeId catalog_type_name,
        in ParameterList additional_parameters
    );

    TransactionalSession current\_session();

    TransactionalSessionList sessions(
```

```
        in CosTransactions::Coordinator transaction
    );

StorageObjectFactory register\_storage\_object\_factory(
    in TypeId storage_type_name,
    in StorageObjectFactory storage_object_factory
);

StorageHomeFactory register\_storage\_home\_factory(
    in TypeId storage_home_type_name,
    in StorageHomeFactory storage_home_factory
);

SessionFactory register\_session\_factory(
    in TypeId catalog_type_name,
    in SessionFactory session_factory
);

SessionPoolFactory register\_session\_pool\_factory(
    in TypeId catalog_type_name,
    in SessionPoolFactory session_pool_factory
);

};
```

Connector::create_basic_session()

```
// PSDL code
Session create_basic_session(
    in AccessMode access_mode,
    in TypeId catalog_type_name,
    in ParameterList additional_parameters
);
```

Creates a basic, non-transactional session and returns a reference to the session.

Parameters

<code>access_mode</code>	The access can be read-only or both read and write.
<code>catalog_type_name</code>	The value is either an empty string or the PSDL type id of a catalog. For example: <code>PSDL:com/acme/People:1.0.</code>
<code>additional_parameters</code>	See Table 2 .

Table 2: *Additional PSS Session Creation Parameters*

Parameter Name	Type	Description
<code>to</code>	<code>string</code>	This is a required parameter. Some string that identifies what you connect to. For example with PSS/DB, it will be an environment name; with PSS/ODBC a datasource name; with PSS/Oracle, an Oracle database name.
<code>concurrent</code>	<code>boolean</code>	Will this session be used by multiple concurrent threads? This parameter is not required. The default value is false.
<code>single writer</code>	<code>boolean</code>	Is this session the only session that writes to this database? When true, there is no risk of deadlock and the cache can be kept as-is after a commit. This parameter is not required. The default value is false.
<code>replicas</code>	<code>IT_PSS::DynamicReplicaSeq</code>	A sequence of <code>IT_PSS::DynamicReplica</code> references, which represents the list of currently active replicas.

Additional Relational Parameters

<code>pessimistic locking</code>	<code>boolean</code>	Does this session acquire a write lock before updating an object in its cache? The default value is true. This parameter is not required.
----------------------------------	----------------------	---

Table 2: *Additional PSS Session Creation Parameters*

Parameter Name	Type	Description
incarnation map size	long	The size of the per-session hash map in which PSS/R keeps incarnations. The given value is rounded up to the closest power of 2. The default value is 1024. This parameter is not required.

Exceptions

`PERSIST_STORE` A session cannot be provided with the desired (or higher) access mode.

See Also

[CosPersistentState::Connector::create_transactional_session\(\)](#)

Connector::create_transactional_session()

```
// PSDL code
TransactionalSession create_transactional_session(
    in AccessMode access_mode,
    in IsolationLevel default_isolation_level,
    in EndOfAssociationCallback callback,
    in TypeId catalog_type_name,
    in ParameterList additional_parameters
);
```

Creates a new transactional session and returns a reference to the session.

Parameters

<code>access_mode</code>	The access can be read-only or both read and write.
<code>default_isolation_level</code>	The isolation level of resources created by this transactional session.
<code>callback</code>	Your application can be notified when a session is released by PSS by passing in an EndOfAssociationCallback local object.

catalog_type_name	The value is either an empty string or the PSDL type id of a catalog. For example: PSDL:com/acme/People:1.0.
additional_parameters	See Table 3 .

Table 3: *Additional PSS TransactionalSession Creation Parameters*

Parameter Name	Type	Description
to	string	This is a required parameter. Some string that identifies what you connect to. For example with PSS/DB, it will be an environment name; with PSS/ODBC a datasource name; with PSS/Oracle, an Oracle database name.
concurrent	boolean	Will this session be used by multiple concurrent threads? This parameter is not required. The default value is false.
single writer	boolean	Is this session the only session that writes to this database? When true, there is no risk of deadlock and the cache can be kept as-is after a commit. This parameter is not required. The default value is false.
replicas	IT_PSS:: DynamicReplicaSeq	A sequence of IT_PSS:: DynamicReplica references, which represents the list of currently active replicas.

Exceptions

PERSIST_STORE Raised if:

- The session cannot be provided with the desired (or higher) access mode.
- The implementation cannot provide the desired default isolation level.

See Also [CosPersistentState::Connector::create_basic_session\(\)](#)

Connector::current_session()

```
// PSDL code
TransactionalSession current_session();
```

Returns the current transactional session. The operation logically calls [sessions\(\)](#) with the transaction associated with the calling thread.

Exceptions

PERSIST_STORE A single session cannot be returned.

See Also [CosPersistentState::Connector::sessions\(\)](#)

Connector::get_pid()

```
// PSDL code
Pid get_pid(
    in StorageObjectBase obj
);
```

Returns the [Pid](#) of the given storage object.

See Also [CosPersistentState::Connector::get_short_pid\(\)](#)

Connector::get_short_pid()

```
// PSDL code
ShortPid get_short_pid(
    in StorageObjectBase obj
);
```

Returns the [ShortPid](#) of the given storage object.

See Also [CosPersistentState::Connector::get_pid\(\)](#)

Connector::implementation_id Attribute

```
// PSDL code
readonly attribute string implementation_id;
```

Returns the Id of this implementation.

Connector::register_session_factory()

```
// PSDL code
SessionFactory register_session_factory(
    in TypeId catalog_type_name,
    in SessionFactory session_factory
);
```

Registers a session factory and returns the factory previously registered with the given name. The operation returns NULL when there is no previously registered factory.

See Also

[CosPersistentState::Connector::register_storage_object_factory\(\)](#)
[CosPersistentState::Connector::register_storage_home_factory\(\)](#)
[CosPersistentState::Connector::register_session_pool_factory\(\)](#)

Connector::register_session_pool_factory()

```
// PSDL code
SessionPoolFactory register_session_pool_factory(
    in TypeId catalog_type_name,
    in SessionPoolFactory session_pool_factory
);
```

Registers session pool factories and returns the factory previously registered with the given name. The operation returns NULL when there is no previously registered factory.

See Also

[CosPersistentState::Connector::register_storage_object_factory\(\)](#)
[CosPersistentState::Connector::register_storage_home_factory\(\)](#)
[CosPersistentState::Connector::register_session_factory\(\)](#)

Connector::register_storage_home_factory()

```
// PSDL code
StorageHomeFactory register_storage_home_factory(
    in TypeId storage_home_type_name,
    in StorageHomeFactory storage_home_factory
);
```

Registers storage home factories and returns the factory previously registered with the given name. The operation returns NULL when there is no previously registered factory.

See Also

[CosPersistentState::Connector::register_storage_object_factory\(\)](#)
[CosPersistentState::Connector::register_session_factory\(\)](#)
[CosPersistentState::Connector::register_session_pool_factory\(\)](#)

Connector::register_storage_object_factory()

```
// PSDL code
StorageObjectFactory register_storage_object_factory(
    in TypeId storage_type_name,
    in StorageObjectFactory storage_object_factory
);
```

Registers storage object factories and returns the factory previously registered with the given name. The operation returns NULL when there is no previously registered factory.

See Also

[CosPersistentState::Connector::register_storage_home_factory\(\)](#)
[CosPersistentState::Connector::register_session_factory\(\)](#)
[CosPersistentState::Connector::register_session_pool_factory\(\)](#)

Connector::sessions()

```
// PSDL code
TransactionalSessionList sessions(
    in CosTransactions::Coordinator transaction
);
```

Returns all the transactional sessions created by this connector that are associated with resources registered with the given transaction. Very often `sessions()` returns a single session.

See Also [CosPersistentState::Connector::current_session\(\)](#)

CosPersistentState_Factory Template

The `CosPersistentState_Factory` class is a helper template you use to build [StorageHomeFactory](#) and [StorageObjectFactory](#) objects. The class contains the following virtual methods.

```
template <class T>
class CosPersistentState_Factory {
public:

    virtual T* create()
        throw(CORBA::SystemException) = 0;

    virtual void _add_ref() {}

    virtual void _remove_ref() {}

    virtual ~CosPersistentState_Factory() {}
};
```


CosPersistentState:: EndOfAssociationCallback Interface

The `EndOfAssociationCallback` interface is implemented by the developer of the application. When a session-resource association has ended, the session may not become available immediately. For example, if the session is implemented using an ODBC or JDBC connection, PSS needs this connection until the resource (ODBC/JDBC transaction) is committed or rolled back.

```
// PSDL code in module CosPersistentState
local interface EndOfAssociationCallback {
    void released(in TransactionalSession session);
};
```

See Also

[CosPersistentState::Connector::create_transactional_session\(\)](#)

CosPersistentState::Session Interface

A PSS session is a logical connection between a process and one or more datastores. There are two kinds of sessions:

- Basic sessions for file-like access.
- Transactional sessions for transactional access. (See the `TransactionalSession` interface.)

You create a basic session by calling `Connector::create_basic_session()`. A basic session is a local object that supports the following interface:

```
// PSDL Code in module CosPersistentState
local interface Session : CatalogBase {};
```

See Also

[IT_PSS::Session](#)

CosPersistentState:: StorageHomeBase Interface

A storage home can have behavior that is described by operations on its abstract storage home(s). An abstract storage home can also define any number of keys; each key declaration implicitly declares a pair of finder operations. All storage home instances implement the local interface `StorageHomeBase`:

```
// PSDL in module CosPersistentState
local interface StorageHomeBase {

    StorageObjectBase find\_by\_short\_pid(
        in ShortPid short_pid
    )
    raises (NotFound);

    CatalogBase get\_catalog();
};
```

StorageHomeBase::find_by_short_pid()

```
// PSDL code
StorageObjectBase find_by_short_pid(
    in ShortPid short_pid
)
    raises (NotFound);
```

Returns a storage object for the given short pid.

Parameters

`short_pid` The short pid in the target storage home.

Exceptions

[CosPersistentS](#) The object is not found.

[tate::](#)

[NotFound](#)

StorageHomeBase::get_catalog()

```
// PSDL code
CatalogBase get_catalog();
```

Returns the catalog that manages the target storage home instance.

CosPersistentState:: StorageHomeFactory Native Type

The `StorageHomeFactory` is a native PSDL type.

```
// PSDL in module CosPersistentState  
native StorageHomeFactory;
```

The C++ mapping of this native type is as follows:

```
// C++  
typedef CosPersistentState_Factory<StorageHomeBase>  
    StorageHomeFactory;
```

The application developer derives a class from this `StorageHomeFactory` type to provide an implementation.

See Also

[IT_PSS_StorageHomeFactory](#)
[CosPersistentState::CosPersistentState_Factory](#)

CosPersistentState::StorageObject Interface

The StorageObject interface supports a PSS storage object.

```
// PSDL in module CosPersistentState
abstract storagetype StorageObject {
    void destroy\_object\(\);
    boolean object\_exists\(\);
    Pid get\_pid\(\);
    ShortPid get\_short\_pid\(\);
    StorageHomeBase get\_storage\_home\(\);
};
```

StorageObject::destroy_object()

```
// PSDL code
void destroy_object();
```

When called on an incarnation, the operation destroys the associated storage object (but does not destroy any of its incarnation).

Exceptions

PERSIST_STORE The operation is called on the instance of an embedded storage object.

StorageObject::get_pid()

```
// PSDL code
Pid get_pid();
```

Returns the [Pid](#) of the associated storage object when called on an incarnation.

Exceptions

`PERSIST_STORE` The operation is called on the instance of an embedded storage object.

See Also

[CosPersistentState::StorageObject::get_short_pid\(\)](#)

StorageObject::get_short_pid()

```
// PSDL code  
ShortPid get_short_pid();
```

Returns the [ShortPid](#) of the associated storage object when called on an incarnation.

Exceptions

`PERSIST_STORE` The operation is called on the instance of an embedded storage object.

See Also

[CosPersistentState::StorageObject::get_pid\(\)](#)

StorageObject::get_storage_home()

```
// PSDL code  
StorageHomeBase get_storage_home();
```

Returns the storage home instance that manages the target storage object instance.

StorageObject::object_exists()

```
// PSDL code  
boolean object_exists();
```

Returns true if the target incarnation represents an actual storage object and false if it does not.

CosPersistentState:: StorageObjectBase Native Type

A storage object can have both state and behavior. The visible part of its state is described by state members on its abstract storage type(s). Similarly, its behavior is described by operations on its abstract storage type(s).

All storage object instances are derived from this common base,
StorageObjectBase:

```
// PSDL in module CosPersistentState  
native StorageObjectBase;
```

The C++ mapping of this native type is as follows:

```
class StorageObjectBase {  
protected:  
    virtual ~StorageObjectBase() {}  
};
```


CosPersistentState:: StorageObjectFactory Native Type

StorageObjectFactory is a native type.

```
// in module CosPersistentState  
native StorageObjectFactory;
```

The C++ mapping of this native type is as follows:

```
// C++  
typedef CosPersistentState_Factory<StorageObject>  
    StorageObjectFactory;
```

The application developer derives a class from this StorageObjectFactory type to provide an implementation.

See Also

[IT_PSS_StorageObjectFactory](#)
[CosPersistentState::CosPersistentState_Factory](#)

CosPersistentState::StorageObjectRef Class

The StorageObjectRef class is a standard C++ base class mapping for a StorageObject reference.

```
class StorageObjectRef {
public:
    typedef StorageObject target_type;

    static CORBA::TypeCode_ptr static_type();

    StorageObjectRef(
        StorageObject* obj = 0,
        CatalogBase_ptr catalog = 0,
        void* impl_data = 0
    );
    StorageObjectRef(
        const StorageObjectRef& ref
    );

    StorageObjectRef& operator=(
        const StorageObjectRef& ref
    );

    StorageObjectRef& operator=(
        StorageObject* obj
    );

    void release();

    StorageObject* operator->(); // not const!

    CORBA::Boolean same_ref(
        StorageObjectRef
    ) const;

    void destroy_object() const;
```

```
Pid* get\_pid\(\) const;

ShortPid* get\_short\_pid\(\) const;

CORBA::Boolean is\_null\(\) const;

StorageHomeBase_ptr get\_storage\_home\(\) const;

// read-only access to data members

void* \_impl\_data\(\) const;

CosPersistentState::CatalogBase_ptr \_catalog\(\) const;

StorageObject* \_target\(\) const;

protected:
    CosPersistentState::CatalogBase_ptr m_catalog;
    void* m_impl_data;
    StorageObject* m_target;
};
```

StorageObjectRef::_catalog()

[CosPersistentState::CatalogBase_ptr _catalog\(\)](#) const;

Returns the catalog of the object.

StorageObjectRef::destroy_object()

void [destroy_object\(\)](#) const;

Destroys the target object.

StorageObjectRef::get_pid()

[Pid](#)* get_pid() const;

Returns the [Pid](#) of the target object.

StorageObjectRef::get_short_pid()

[ShortPid](#)* get_short_pid() const;

Returns the short pid of the target object.

StorageObjectRef::get_storage_home()

[StorageHomeBase_ptr](#) get_storage_home() const;

Returns the storage home of the target object.

StorageObjectRef::_impl_data()

void* _impl_data() const;

StorageObjectRef::is_null()

[CORBA::Boolean](#) is_null() const;

Returns true if and only if this reference is null.

StorageObjectRef::operator=()

```
StorageObjectRef& operator=(  
    const StorageObjectRef& ref  
);
```

An assignment operator that takes an incarnation of the target abstract storage type.

```
StorageObjectRef& operator=(  
    StorageObject* obj  
);
```

An assignment operator.

StorageObjectRef::operator->()

```
StorageObject* operator->(); // not const!
```

A de-reference operator that de-references this reference and returns the target object. The caller is not supposed to release this incarnation.

StorageObjectRef::release()

```
void release();
```

Releases the reference.

StorageObjectRef::same_ref()

```
CORBA::Boolean same_ref(  
    StorageObjectRef  
) const;
```

Returns true if the input storage object reference is the same as this one.

StorageObjectRef::_static_type()

```
static CORBA::TypeCode\_ptr _static_type();
```

Returns a `TypeCode` reference.

StorageObjectRef::StorageObjectRef() Constructors

```
StorageObjectRef(  
    StorageObject* obj      = 0,  
    CatalogBase_ptr catalog = 0,
```

```
        void*          impl_data = 0
    );
```

The default constructor creates a null reference.

```
StorageObjectRef(
    const StorageObjectRef& ref
);
```

A non-explicit constructor that takes an incarnation of the target abstract storage type.

StorageObjectRef::_target_type

```
typedef StorageObject _target_type;
```

A type definition to the target type. This is useful for programming with templates.

StorageObjectRef::_target()

```
StorageObject* _target() const;
```

Returns the target object.

CosPersistentState:: TransactionalSession Interface

A transactional session is a specialized session that provides transactional access to storage objects. A transactional session supports the local interface `TransactionalSession`.

At a given time, a transactional session can be associated with one resource object (a datastore transaction), or with no resource at all. The session-resource association can be active, suspended, or ending. The state members of an incarnation managed by a transactional session can be used only when this session has an active association with a resource.

Typically, a resource is associated with a single session for its entire lifetime. However, with some advanced database products, the same resource may be associated with several sessions, possibly at the same time.

You create a transaction session by calling [create_transactional_session\(\)](#).

```
// PSDL Code in module CosPersistentState
local interface TransactionalSession : Session {

    readonly attribute IsolationLevel default\_isolation\_level;

    typedef short AssociationStatus;
    const AssociationStatus NO_ASSOCIATION = 0;
    const AssociationStatus ACTIVE         = 1;
    const AssociationStatus SUSPENDED     = 2;
    const AssociationStatus ENDING        = 3;

    void start(in CosTransactions::Coordinator transaction);
    void suspend(in CosTransactions::Coordinator transaction);
    void end(
        in CosTransactions::Coordinator transaction,
        in boolean success
    );
};
```

```
AssociationStatus get\_association\_status\(\);  
  
CosTransactions::Coordinator transaction\(\);  
};
```

See Also

[IT_PSS::TransactionalSession](#)
[CosPersistentState::Session](#)

TransactionalSession::AssociationStatus Type

```
// PSDL Code  
typedef short AssociationStatus;  
const AssociationStatus NO_ASSOCIATION = 0;  
const AssociationStatus ACTIVE = 1;  
const AssociationStatus SUSPENDED = 2;  
const AssociationStatus ENDING = 3;
```

The association status of a resource with a session. Valid values include:

```
NO_ASSOCIATION  
ACTIVE  
SUSPENDED  
ENDING
```

See Also

[CosPersistentState::TransactionalSession::
get_association_status\(\)](#)

TransactionalSession::default_isolation_level Attribute

```
// PSDL Code  
readonly attribute IsolationLevel default_isolation_level;
```

Returns the default isolation level of resources created for this transactional session.

TransactionalSession::end()

```
// PSDL Code  
void end(  
    in CosTransactions::Coordinator transaction,  
    in boolean success
```

```
);
```

Terminates a session-transaction association.

Parameters

`transaction` The transaction of the resource.

`success` If the `success` parameter is FALSE, the resource is rolled back immediately. Like [refresh\(\)](#), `end()` invalidates direct references to incarnations' data members.

A resource can be prepared or committed in one phase only when it is not actively associated with any session. The resource will rollback if it is asked to prepare or commit in one phase when still in use. A resource ends any session-resource association in which it is involved when it is prepared, committed in one phase, or rolled back.

Exceptions

`PERSIST_STORE` No associated resource.

`INVALID_TRANSACTION` The given transaction does not match the transaction of the resource associated with this session.

The standard exception `is` raised if

See Also

[CosPersistentState::TransactionalSession::start\(\)](#)
[CosPersistentState::TransactionalSession::suspend\(\)](#)

TransactionalSession::get_association_status()

```
// PSDL Code  
AssociationStatus get_association_status();
```

Returns the status of the association (if any) with this session.

TransactionalSession::start()

```
// PSDL Code  
void start(  
    in CosTransactions::Coordinator transaction  
);
```

Starts the transaction.

Parameters

`transaction` The transaction to start.

This operation does one of three things depending on the association of the transaction:

1. When `transaction` matches the transaction of the suspended (or ending) association, `start()` re-activates a suspended (or ending) session-resource association.
2. If a resource compatible with this session is already associated with the given transaction, `start()` associates this resource with this session, and makes the association active.
3. If the session creates a new resource and registers it with the given transaction. The session also associates itself with this resource and makes the association active.

Exceptions

`INVALID_TRANSACTION` There is a suspended (or ending) association but the transactions do not match.

See Also

[CosPersistentState::TransactionalSession::suspend\(\)](#)
[CosPersistentState::TransactionalSession::end\(\)](#)

TransactionalSession::suspend()

```
// PSDL Code
void suspend(
    in CosTransactions::Coordinator transaction
);
```

Suspends a session-resource association.

Parameters

`transaction` The transaction to suspend.

Exceptions

`PERSIST_STORE` No active association.

The standard exception `INVALID_TRANSACTION` is raised if the given transaction does not match the transaction of the resource actively associated with this session.

See Also

[CosPersistentState::TransactionalSession::start\(\)](#)
[CosPersistentState::TransactionalSession::end\(\)](#)

TransactionalSession::transaction()

```
// PSDL Code  
CosTransactions::Coordinator transaction();
```

Returns the coordinator of the transaction with which the resource associated with this session is registered. The operation returns a nil object reference when the session is not associated with a resource.

IT_PSS Overview

The IT_PSS interfaces consist of:

[CatalogBase](#)
[Connector](#)
[DynamicReplica](#)
[Master](#)
[PreparedStatement](#)
[Replica](#)
[ResultSet](#)
[Session](#)
[SessionManager](#)
[Statement](#)
[StorageObject](#)
[TransactionalSession](#)
[TransactionalSession2](#)

This module also has the following helper classes:

[IT_PSS_StorageHomeFactory](#)
[IT_PSS_StorageObjectFactory](#)
[TxSessionAssociation](#)

IT_PSS::CatalogBase Interface

PSS provides simple JDBC-like queries. You use `CatalogBase` to create a [Statement](#) or [PreparedStatement](#). The query language is a subset of SQL that currently only supports the following form of select query:

```
select ref(h) from home_type_id h
```

The PSDL code is as follows:

```
// PSDL Code in Module IT_PSS
local interface CatalogBase :
CosPersistentState::CatalogBase {

    Statement it\_create\_statement\(\);

    Statement it\_create\_statement\_with\_type\_and\_concurrency(
        in ResultSet::Type type,
        in ResultSet::Concurrency concurrency
    );

    PreparedStatement it\_prepare\_statement(
        in string pssql
    );

    PreparedStatement
it\_prepare\_it\_statement\_with\_type\_and\_concurrency(
        in string pssql,
        in ResultSet::Type type,
        in ResultSet::Concurrency concurrency
    );

    void it\_discard\_flush\_list\(\);

    void it\_discard\_all(
        in boolean clear_non_id_refs
    );
};
```

Enhancement This is an Orbix enhancement.

See Also

[CosPersistentState::CatalogBase](#)
[IT_PSS::PreparedStatement](#)
[IT_PSS::Statement](#)
[IT_PSS::ResultSet](#)

CatalogBase::it_create_statement()

```
// PSDL Code  
Statement it_create_statement();
```

Creates and returns a JDBC-like [Statement](#).

Enhancement This is an Orbix enhancement.

CatalogBase:: it_create_statement_with_type_and_concurrency()

```
// PSDL Code  
Statement it_create_statement_with_type_and_concurrency(  
    in ResultSet::Type type,  
    in ResultSet::Concurrency concurrency  
);
```

Creates and returns a JDBC-like [Statement](#) with a specific [ResultSet](#) type. The concurrency setting can be either read-only or updateable. Only one [ResultSet](#) per [Statement](#) can be open at any point in time. All statement execute methods implicitly close a statement's current [ResultSet](#) if an open one exists.

Enhancement This is an Orbix enhancement.

CatalogBase::it_discard_all()

```
// PSDL Code  
void it_discard_all(  
    in boolean clear_non_id_refs  
);
```

Discards all cached objects.

Parameters

`clear_non_id_refs` If this parameter is set to true, any references that an object might have to another object are removed. This removes the possibility of circular references between objects.

Enhancement This is an Orbix enhancement.

CatalogBase::it_discard_flush_list()

```
// PSDL Code
void it_discard_flush_list();
```

Discards all modified objects in the catalog.

Enhancement This is an Orbix enhancement.

CatalogBase::it_prepare_statement()

```
// PSDL Code
PreparedStatement it_prepare_statement(
    in string pssql
);
```

Creates and returns a JDBC-like [PreparedStatement](#) with the given query.

Enhancement This is an Orbix enhancement.

CatalogBase:: it_prepare_statement_with_type_and_concurrency()

```
// PSDL Code
PreparedStatement it_prepare_statement_with_type_and_concurrency(
    in string pssql,
    in ResultSet::Type type,
    in ResultSet::Concurrency concurrency
);
```

Creates and returns a JDBC-like [PreparedStatement](#) with a given query and specific `ResultSet` type. The concurrency setting can be either read-only or updateable.

Enhancement This is an Orbix enhancement.

IT_PSS::Connector Interface

This is an Orbix-enhancement interface that lets you create a session manager.

```
// PSDL Code in module IT_PSS
/* local */ interface Connector : CosPersistentState::Connector {
    SessionManager it\_create\_session\_manager(
        in CosPersistentState::ParameterList parameters
    );
};
```

Enhancement This is an Orbix enhancement.

See Also [CosPersistentState::Connector](#)

Connector::it_create_session_manager()

```
// PSDL Code
SessionManager it_create_session_manager(
    in CosPersistentState::ParameterList parameters
);
```

Creates and returns a session manager.

Parameters

`parameters` See [Table 4](#) for details about possible parameters. Other parameters are passed in each session creation call. You cannot, however, pass a parameter named `concurrent` when creating a session manager. The session manager's read-only read-committed session is created with `concurrent` set to true, whereas the session manager's read-write serializable sessions are created with `concurrent` set to false.

Table 4: *Additional PSS SessionManager Creation Parameters*

Parameter Name	Type	Description
to	string	This parameter is required. Some string that identifies what you connect to. For example with PSS/DB, it will be an environment name; with PSS/ODBC a datasource name; with PSS/Oracle, an Oracle database name.
rw pool size	long	Initial size of the pool of read-write transactional sessions managed by the session manager. Must be between 1 and 1000. This parameter is not required. The default value is 1.
grow pool	boolean	Create a new session to process a new request when all the read-write transactional sessions are busy? If false, wait until a read-write transactional session becomes available. This parameter is not required. The default value is false.
single writer	boolean	Can be true only when rw pool size is 1, in which case the read-write transactional session will be created with the single writer parameter set to true. This parameter is not required. The default value is false.
replicas	IT_PSS::DynamicReplicaSeq	A sequence of IT_PSS::DynamicReplica references, which represents the list of currently active replicas.

Enhancement This is an Orbix enhancement.

See Also [IT_PSS::SessionManager](#)

IT_PSS::DynamicReplica Interface

The `DynamicReplica` interface provides functionality for replicated databases. Since Orbix 6.2, the `DynamicReplica` interface replaces the functionality of both the `IT_PSS::Master` and `IT_PSS::Replica` interfaces. The inherited operations are now deprecated.

```
interface DynamicReplica : Master, Replica
{ };
```


IT_PSS:Master Interface

Note: The `Master` interface is deprecated since Orbix 6.2. You should now use the `IT_PSS::DynamicReplica` type to hold a reference to a replica (for backwards compatibility, however, the `Replica` interface is still supported).

The `Master` interface provides functionality for master instances of replicated persistent objects using the persistent state service.

```
interface Master
{}
```


IT_PSS::PreparedStatement Interface

The `PreparedStatement` interface is a JDBC-like prepared statement which is an object that represents a pre-compiled SQL statement. An SQL statement is pre-compiled and stored in the `PreparedStatement` object so your application can then efficiently execute the statement multiple times.

```
// PSDL Code in module IT_PSS
local interface PreparedStatement : Statement {

    void execute\_prepared\(\);

    ResultSet execute\_prepared\_query\(\);

    unsigned long execute\_prepared\_update\(\);

    void define\_parameter(
        in unsigned short parameter_index,
        in any parameter_value
    );

    void clear\_parameters\(\);
};
```

Enhancement This is an Orbix enhancement.

See Also [IT_PSS::CatalogBase](#)
[IT_PSS::Statement](#)

PreparedStatement::clear_parameters()

```
// PSDL Code
void clear_parameters();
```

Clears the current parameter values immediately.

Enhancement This is an Orbix enhancement.

PreparedStatement::define_parameter()

```
// PSDL Code
void define_parameter(
    in unsigned short parameter_index,
    in any parameter_value
);
```

Defines an SQL parameter value for the designated parameter index.

Enhancement This is an Orbix enhancement.

PreparedStatement::execute_prepared()

```
// PSDL Code
void execute_prepared();
```

Executes the prepared SQL statement.

Enhancement This is an Orbix enhancement.

See Also

[IT_PSS::PreparedStatement::execute_prepared_query\(\)](#)
[IT_PSS::PreparedStatement::execute_prepared_update\(\)](#)

PreparedStatement::execute_prepared_query()

```
// PSDL Code
ResultSet execute_prepared_query();
```

Executes the SQL query in this `PreparedStatement` object and returns the result set generated by the query.

Enhancement This is an Orbix enhancement.

See Also

[IT_PSS::PreparedStatement::execute_prepared\(\)](#)
[IT_PSS::PreparedStatement::execute_prepared_update\(\)](#)

PreparedStatement::execute_prepared_update()

```
// PSDL Code
unsigned long execute_prepared_update();
```

Executes the SQL INSERT, UPDATE or DELETE statement in this PreparedStatement object.

Enhancement This is an Orbix enhancement.

See Also

[IT_PSS::PreparedStatement::execute_prepared\(\)](#)
[IT_PSS::PreparedStatement::execute_prepared_query\(\)](#)

IT_PSS:Replica Interface

Note: The `Replica` interface is deprecated since Orbix 6.2. You should now use the `IT_PSS::DynamicReplica` type to hold a reference to a replica (for backwards compatibility, however, the `Replica` interface is still supported).

The `Replica` interface provides functionality for replicated databases. The persistent state service supports two styles of replicas: a push-style replica and a pull-style replica. A push-style replica is updated by the master instance of the object. A pull-style replica requests updates periodically from the master instance of the object.

```
interface Replica
{
    boolean set_master(in Master new_master);

    readonly attribute unsigned long long last_successful_refresh;

    // Pull refresh now
    void refresh();
};
```

IT_PSS::Replica::set_master

```
boolean set_master(in Master new_master)
```

Registers the replica with a master instance of the object. It returns `TRUE` if the registration is successful.

Parameters

This function takes an object of [Master](#) containing an object reference to the master instance of the object.

IT_PSS::Replica::last_successful_refresh

readonly attribute unsigned long long last_successful_refresh

Returns the amount of time that has passed since the last time the replica was successfully refreshed by the master instance of the object.

IT_PSS:Replica:refresh

void refresh()

Requests an update from the master instance of the object. The master will completely sync the replica as a result of this call.

IT_PSS::ResultSet Interface

The `ResultSet` interface provides access to a table of data similar to a JDBC result set. A `ResultSet` object is usually generated by executing a [Statement](#) or a [PreparedStatement](#). A `ResultSet` maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row.

Data types include:

[Concurrency](#) Type
[FetchDirection](#) Type
[Type](#)

Operations include:

absolute()	get_fetch_size()	next()
after_last()	get_row()	previous()
before_first()	get_statement()	refresh_row()
cancel_row_updates()	get_type()	relative()
close()	insert_row()	row_deleted()
delete_row()	is_after_last()	row_inserted()
find_state_member()	is_before_first()	row_updated()
first()	is_first()	set()
get()	is_last()	set_by_name()
get_by_name()	last()	set_fetch_direction()
get_concurrency()	move_to_current_row()	set_fetch_size()
get_fetch_direction()	move_to_insert_row()	update_row()

Enhancement This interface is an Orbix enhancement.

See Also [IT_PSS::CatalogBase](#)

```
// PSDL Code in module IT_PSS
local interface ResultSet {

    typedef unsigned short Type;
    const Type TYPE_FORWARD_ONLY = 1;
    const Type TYPE_SCROLL_INSENSITIVE = 2;
    const Type TYPE_SCROLL_SENSITIVE = 3;
```

```
typedef unsigned short Concurrency;
const Concurrency CONCUR_READ_ONLY = 1;
const Concurrency CONCUR_UPDATABLE = 2;

typedef unsigned short FetchDirection;
const FetchDirection FETCH_FORWARD = 1;
const FetchDirection FETCH_REVERSE = 2;
const FetchDirection FETCH_UNKNOWN = 3;

Statement get\_statement\(\);

// Basic operations
//
boolean next\(\);
void close\(\);

any get(
    in unsigned short index
);

any get\_by\_name(
    in string state_member_name
);

// Find state_member
//
unsigned short find\_state\_member(
    in string state_member_name
);

// Getting/setting the current row
//
boolean is\_after\_last\(\);
boolean is\_before\_first\(\);
boolean is\_first\(\);
boolean is\_last\(\);
void after\_last\(\);
void before\_first\(\);
boolean first\(\);
boolean last\(\);
unsigned short get\_row\(\);

boolean absolute(
```

```
        in short row
    );

    boolean relative(
        in short rows
    );

    boolean previous\(\);
    void move\_to\_insert\_row\(\);
    void move\_to\_current\_row\(\);

    // Fetch direction and size
    //
    void set\_fetch\_direction(
        in FetchDirection direction
    );

    FetchDirection get\_fetch\_direction\(\);

    void set\_fetch\_size(
        in unsigned short fetch_size
    );

    unsigned short get\_fetch\_size\(\);

    // Type and Concurrency
    //
    Type get\_type\(\);
    Concurrency get\_concurrency\(\);

    // Was row modified?
    //
    boolean row\_updated\(\);
    boolean row\_inserted\(\);
    boolean row\_deleted\(\);

    // Write operations
    //
    void set(
        in unsigned short index,
        in any value
    );
```

```
void set\_by\_name(
    in string state_member_name,
    in any value
);

void insert\_row();
void update\_row();
void delete\_row();
void refresh\_row();
void cancel\_row\_updates();
};
```

ResultSet::absolute()

```
// PSDL Code
boolean absolute(
    in short row
);
```

Moves the cursor to the given row number in the result set.

Parameters

row If the row number is positive, the cursor moves to the given row number with respect to the beginning of the result set. The first row is row 1, the second is row 2, and so on.

 If the given row number is negative, the cursor moves to an absolute row position with respect to the end of the result set. For example, calling `absolute(-1)` positions the cursor on the last row, `absolute(-2)` indicates the next-to-last row, and so on.

 An attempt to position the cursor beyond the first/last row in the result set leaves the cursor before/after the first/last row, respectively.

Enhancement This is an Orbix enhancement.

ResultSet::after_last()

```
// PSDL Code
void after_last();
```

Moves the cursor to the end of the result set, just after the last row. Has no effect if the result set contains no rows.

Enhancement This is an Orbix enhancement.

ResultSet::before_first()

```
// PSDL Code
void before_first();
```

Moves the cursor to the front of the result set, just before the first row. Has no effect if the result set contains no rows.

Enhancement This is an Orbix enhancement.

ResultSet::cancel_row_updates()

```
// PSDL Code
void cancel_row_updates();
```

Cancels the updates made to a row in the table.

Enhancement This is an Orbix enhancement.

ResultSet::close()

```
// PSDL Code
void close();
```

Releases this `ResultSet` object's database and JDBC resources immediately instead of waiting for this to happen when it is automatically closed.

Enhancement This is an Orbix enhancement.

ResultSet::Concurrency Type

```
// PSDL Code
typedef unsigned short Concurrency;
const Concurrency CONCUR_READ_ONLY = 1;
const Concurrency CONCUR_UPDATABLE = 2;
```

The concurrency mode of the table. It can be read-only or updated.

Enhancement This is an Orbix enhancement.

ResultSet::delete_row()

```
// PSDL Code
void delete_row();
```

Deletes the current row from the table.

Enhancement This is an Orbix enhancement.

ResultSet::FetchDirection Type

```
// PSDL Code
typedef unsigned short FetchDirection;
const FetchDirection FETCH_FORWARD = 1;
const FetchDirection FETCH_REVERSE = 2;
const FetchDirection FETCH_UNKNOWN = 3;
```

Defines the direction of table row processing.

FETCH_FORWARD	The rows in a result set will be processed in a forward direction; first-to-last.
FETCH_REVERSE	The rows in a result set will be processed in a reverse direction; last-to-first.
FETCH_UNKNOWN	The order in which rows in a result set will be processed is unknown.

Enhancement This is an Orbix enhancement.

ResultSet::find_state_member()

```
// PSDL Code
unsigned short find_state_member(
    in string state_member_name
);
```

Returns the index for the given result set's state member name.

Enhancement This is an Orbix enhancement.

ResultSet::first()

```
// PSDL Code  
boolean first();
```

Moves the cursor to the first row in the result set. Returns true if the cursor is on a valid row; false if there are no rows in the result set

Enhancement This is an Orbix enhancement.

ResultSet::get()

```
// PSDL Code  
any get(  
    in unsigned short index  
);
```

Returns the value for the given parameter index.

Enhancement This is an Orbix enhancement.

See Also [IT_PSS::ResultSet::set\(\)](#)

ResultSet::get_by_name()

```
// PSDL Code  
any get_by_name(  
    in string state_member_name  
);
```

Returns the value for a state member given the member name.

Enhancement This is an Orbix enhancement.

See Also [IT_PSS::ResultSet::set_by_name\(\)](#)

ResultSet::get_concurrency()

```
// PSDL Code  
Concurrency get_concurrency();
```

Returns the concurrency value.

Enhancement This is an Orbix enhancement.

ResultSet::get_fetch_direction()

```
// PSDL Code  
FetchDirection get_fetch_direction();
```

Returns the direction of table row processing.

Enhancement This is an Orbix enhancement.

See Also [IT_PSS::ResultSet::set_fetch_direction\(\)](#)

ResultSet::get_fetch_size()

```
// PSDL Code  
unsigned short get_fetch_size();
```

Returns the number of rows that are fetched from the database when more rows are needed for this result set.

Enhancement This is an Orbix enhancement.

See Also [IT_PSS::ResultSet::set_fetch_size\(\)](#)

ResultSet::get_row()

```
// PSDL Code  
unsigned short get_row();
```

Returns the current row number. The first row is number 1, the second number is 2, and so on.

Enhancement This is an Orbix enhancement.

ResultSet::get_statement()

```
// PSDL Code
Statement get_statement();
```

Returns the [Statement](#) that produced this `ResultSet` object.

Enhancement This is an Orbix enhancement.

ResultSet::get_type()

```
// PSDL Code
Type get_type();
```

Returns the type of this result set. The type is determined by the `Statement` that created the result set.

Enhancement This is an Orbix enhancement.

ResultSet::insert_row()

```
// PSDL Code
void insert_row();
```

Inserts a row.

Enhancement This is an Orbix enhancement.

ResultSet::is_after_last()

```
// PSDL Code
boolean is_after_last();
```

Returns true if the cursor is after the last row in the result set, false if it is not.

Enhancement This is an Orbix enhancement.

ResultSet::is_before_first()

```
// PSDL Code
boolean is_before_first();
```

Returns true if the cursor is before the first row in the result set, false if it is not.

Enhancement This is an Orbix enhancement.

ResultSet::is_first()

```
// PSDL Code  
boolean is_first();
```

Returns true if the cursor is on the first row of the result set, false if it is not.

Enhancement This is an Orbix enhancement.

ResultSet::is_last()

```
// PSDL Code  
boolean is_last();
```

Returns true if the cursor is on the last row of the result set, false if it is not.

Enhancement This is an Orbix enhancement.

ResultSet::last()

```
// PSDL Code  
boolean last();
```

Moves the cursor to the last row in the result set and returns true if the cursor is on a valid row; false if there are no rows in the result set.

Enhancement This is an Orbix enhancement.

ResultSet::move_to_current_row()

```
// PSDL Code  
void move_to_current_row();
```

Moves the cursor to the remembered cursor position, usually the current row. This operation has no effect if the cursor is not on the insert row.

Enhancement This is an Orbix enhancement.

ResultSet::move_to_insert_row()

```
// PSDL Code  
void move_to_insert_row();
```

Moves the cursor to the insert row.

Enhancement This is an Orbix enhancement.

ResultSet::next()

```
// PSDL Code  
boolean next();
```

Moves the cursor down one row from its current position. A ResultSet cursor is initially positioned before the first row; the first call to next makes the first row the current row; the second call makes the second row the current row, and so on.

Enhancement This is an Orbix enhancement.

ResultSet::previous()

```
// PSDL Code  
boolean previous();
```

Moves the cursor to the previous row in the result set.

Enhancement This is an Orbix enhancement.

ResultSet::refresh_row()

```
// PSDL Code  
void refresh_row();
```

Refreshes the current row with its most recent value in the database. This cannot be called when the cursor is on the insert row.

Enhancement This is an Orbix enhancement.

ResultSet::relative()

```
// PSDL Code
boolean relative(
    in short rows
);
```

Moves the cursor a relative number of rows, either positive or negative. Attempting to move beyond the first/last row in the result set positions the cursor before/after the first/last row. Calling `relative(0)` is valid, but does not change the cursor position.

Enhancement This is an Orbix enhancement.

ResultSet::row_deleted()

```
// PSDL Code
boolean row_deleted();
```

Indicates whether a row has been deleted. A deleted row may leave a visible “hole” in a result set. This operation can be used to detect holes in a result set. The value returned depends on whether or not the result set can detect deletions.

Enhancement This is an Orbix enhancement.

ResultSet::row_inserted()

```
// PSDL Code
boolean row_inserted();
```

Indicates whether the current row has had an insertion. The value returned depends on whether or not the result set can detect visible inserts. The operation returns true if a row has had an insertion and insertions are detected.

Enhancement This is an Orbix enhancement.

ResultSet::row_updated()

```
// PSDL Code
boolean row_updated();
```

Indicates whether the current row has been updated. The value returned depends on whether or not the result set can detect updates. If the set can detect updates, the operation returns true if the row has been visibly updated by the owner or another.

Enhancement This is an Orbix enhancement.

ResultSet::set()

```
// PSDL Code
void set(
    in unsigned short index,
    in any value
);
```

Sets the value and parameter index.

Enhancement This is an Orbix enhancement.

See Also [IT_PSS::ResultSet::get\(\)](#)

ResultSet::set_by_name()

```
// PSDL Code
void set_by_name(
    in string state_member_name,
    in any value
);
```

Sets the value for an object's member given the name of the member.

Enhancement This is an Orbix enhancement.

See Also [IT_PSS::ResultSet::get_by_name\(\)](#)

ResultSet::set_fetch_direction()

```
// PSDL Code
void set_fetch_direction(
    in FetchDirection direction
);
```

Sets a hint as to the direction in which the rows in this result set will be processed. The initial value is determined by the statement that produced the result set. The fetch direction may be changed at any time.

Enhancement This is an Orbix enhancement.

See Also [IT_PSS::ResultSet::get_fetch_direction\(\)](#)

ResultSet::set_fetch_size()

```
// PSDL Code
void set_fetch_size(
    in unsigned short fetch_size
);
```

The fetch size is a hint as to the number of rows that should be fetched from the database when more rows are needed for this result set. The default value is set by the [Statement](#) that created the result set. The fetch size may be changed at any time.

Parameters

`fetch_size` If the fetch size is zero, a best guess is used.

Enhancement This is an Orbix enhancement.

See Also [IT_PSS::ResultSet::get_fetch_size\(\)](#)

ResultSet::Type

```
// PSDL Code
typedef unsigned short Type;
const Type TYPE_FORWARD_ONLY = 1;
const Type TYPE_SCROLL_INSENSITIVE = 2;
const Type TYPE_SCROLL_SENSITIVE = 3;
```

The type of this result set. The type is determined by the [Statement](#) that created the result set.

Enhancement This is an Orbix enhancement.

ResultSet::update_row()

```
// PSDL Code  
void update_row();
```

Updates the underlying database with the new contents of the current row.
Cannot be called when the cursor is on the insert row.

Enhancement This is an Orbix enhancement.

IT_PSS::Session Interface

When you create a session with an IONA PSS implementation, you get an `IT_PSS::Session`.

```
// PSDL Code in module IT_PSS
local interface Session : CatalogBase, CosPersistentState::Session
{}
```

Enhancement This interface is an Orbix enhancement.

See Also [IT_PSS::CatalogBase](#)
[CosPersistentState::Session](#)

IT_PSS::SessionManager Interface

PSS fully support transactions, and works with any compliant transaction service implementation. Unless you are developing a trivial demonstration program, you should use transactions when developing applications with PSS.

You can use a `SessionManager` object to manage transactional sessions. A common pattern when developing a transactional server using PSS is to use a shared read-only read-committed transactional session for simple read-only non-transactional requests. Of course, you can also create and manage your transactional sessions directly with the standard lower level PSS APIs from the `CosPersistentState` module.

```
//PSDL in module IT_PSS
local interface SessionManager {
    TransactionalSession get\_shared\_read\_only\_session\_nc\(\);
    void block\_readers\_until\_idle\(\);
};
```

Enhancement This interface is an Orbix enhancement.

See Also [IT_PSS::Connector::it_create_session_manager\(\)](#)

SessionManager::get_shared_read_only_session_nc()

```
//PSDL code
TransactionalSession get_shared_read_only_session_nc();
```

Returns a shared, read-only transactional session. In this context, shared means the transactional session is usable by multiple threads.

Enhancement This is an Orbix enhancement.

SessionManager::block_readers_until_idle()

```
//PSDL code
void block_readers_until_idle();
```

Blocks new threads from using the shared, read-only transactional session until no thread is using the session.

Enhancement This is an Orbix enhancement.

IT_PSS::Statement Interface

The `Statement` interface provides operations for a JDBC-like statement, an object used for executing a static SQL statement and obtaining the results produced by it.

```
// PSDL Code in module IT_PSS
local interface Statement {
    void execute(
        in string pssql
    );

    ResultSet execute\_query(
        in string pssql
    );

    unsigned long execute\_update(
        in string pssql
    );

    ResultSet get\_result\_set();

    void close();

    // Default fetch direction and size
    //
    void set\_fetch\_direction(
        in ResultSet::FetchDirection direction
    );

    ResultSet::FetchDirection get\_fetch\_direction();

    void set\_fetch\_size(
        in unsigned short fetch_size
    );

    unsigned short get\_fetch\_size();

    // Type and Concurrency
```

```
    //
    ResultSet::Type get\_result\_set\_type\(\);
    ResultSet::Concurrency get\_result\_set\_concurrency\(\);
    CatalogBase get\_catalog\(\);
};
```

Enhancement This interface is an Orbix enhancement.

See Also [IT_PSS::CatalogBase](#)
[IT_PSS::PreparedStatement](#)

Statement::close()

```
// PSDL Code
void close();
```

Releases this `Statement` object's database and resources immediately instead of waiting for this to happen when it is automatically closed.

Enhancement This is an Orbix enhancement.

Statement::execute()

```
// PSDL Code
void execute(
    in string pssql
);
```

Executes an SQL statement that may obtain multiple results.

Enhancement This is an Orbix enhancement.

Statement::execute_query()

```
// PSDL Code
ResultSet execute_query(
    in string pssql
);
```

Executes an SQL statement that returns a single [ResultSet](#).

Enhancement This is an Orbix enhancement.

Statement::execute_update()

```
// PSDL Code
unsigned long execute_update(
    in string pssql
);
```

Executes an SQL INSERT, UPDATE or DELETE statement.

Enhancement This is an Orbix enhancement.

Statement::get_catalog()

```
// PSDL Code
CatalogBase get_catalog();
```

Returns the catalog for this Statement.

Enhancement This is an Orbix enhancement.

Statement::get_fetch_direction()

```
// PSDL Code
ResultSet::FetchDirection get_fetch_direction();
```

Returns the direction for fetching rows from database tables that is the default for result sets generated from this Statement object.

Enhancement This is an Orbix enhancement.

Statement::get_fetch_size()

```
// PSDL Code
unsigned short get_fetch_size();
```

Returns the number of result set rows that is the default fetch size for result sets generated from this Statement object.

Enhancement This is an Orbix enhancement.

Statement::get_result_set()

```
// PSDL Code  
ResultSet get_result_set();
```

Returns the current result as a [ResultSet](#) object.

Enhancement This is an Orbix enhancement.

Statement::get_result_set_concurrency()

```
// PSDL Code  
ResultSet::Concurrency get_result_set_concurrency();
```

Returns the result set concurrency.

Enhancement This is an Orbix enhancement.

Statement::get_result_set_type()

```
// PSDL Code  
ResultSet::Type get_result_set_type();
```

Returns the type of the [ResultSet](#).

Enhancement This is an Orbix enhancement.

Statement::set_fetch_direction()

```
// PSDL Code  
void set_fetch_direction(  
    in ResultSet::FetchDirection direction  
);
```

Sets a hint as to the direction in which the rows in a result set should be processed.

Enhancement This is an Orbix enhancement.

Statement::set_fetch_size()

```
// PSDL Code
void set_fetch_size(
    in unsigned short fetch_size
);
```

Gives a hint as to the number of rows that should be fetched from the database when more rows are needed.

Enhancement This is an Orbix enhancement.

IT_PSS_StorageHomeFactory Template

Use this template class to help implement your [StorageHomeFactory](#).

```
template<class T>
class IT_PSS_StorageHomeFactory :
public CosPersistentState::StorageHomeFactory {
    public:

        IT\_PSS\_StorageHomeFactory\(\);

        virtual void \_add\_ref\(\);

        virtual void \_remove\_ref\(\);

        virtual CosPersistentState::StorageHomeBase_ptr create\(\)
            throw(CORBA::SystemException);

    private:
        ...
};
```

Enhancement This is an Orbix enhancement.

IT_PSS_StorageHomeFactory::_add_ref()

```
virtual void _add_ref();
```

Increases the reference count by one.

Enhancement This is an Orbix enhancement.

IT_PSS_StorageHomeFactory::create()

```
virtual CosPersistentState::StorageHomeBase\_ptr create()  
    throw(CORBA::SystemException);
```

Creates and returns a new [StorageHomeBase](#) object.

IT_PSS_StorageHomeFactory::IT_PSS_StorageHomeFactory()

```
IT_PSS_StorageHomeFactory();
```

The constructor.

Enhancement This is an Orbix enhancement.

IT_PSS_StorageHomeFactory::_remove_ref()

```
virtual void _remove_ref();
```

Decreases the reference count by one.

Enhancement This is an Orbix enhancement.

IT_PSS::StorageObject Interface

PSS presents persistent information as storage objects. Each storage object has a type that defines its members and operations. When you create a storage object with an IONA PSS implementation, you get an `IT_PSS::StorageObject`.

```
// PSDL Code in module IT_PSS
abstract storagetype StorageObject {
    void it\_lock\(\);

};
```

Enhancement This interface is an Orbix enhancement.

See Also [CosPersistentState::StorageObject](#)

StorageObject::it_lock()

```
// PSDL Code
void it_lock();
```

This operation acquires an exclusive lock on behalf of a basic session or transactional session.

Enhancement This is an Orbix enhancement.

IT_PSS_StorageObjectFactory Template

Use this template class to help implement your [StorageObjectFactory](#).

```
// c++
template<class T>
class IT_PSS_StorageObjectFactory :
public CosPersistentState::StorageObjectFactory {
public:

    IT\_PSS\_StorageObjectFactory\(\);

    virtual void \_add\_ref\(\);

    virtual void \_remove\_ref\(\);

    virtual CosPersistentState::StorageObject* create\(\)
        throw(CORBA::SystemException);

private:
    ...
};
```

Enhancement This is an Orbix enhancement.

IT_PSS_StorageObjectFactory::_add_ref()

```
virtual void _add_ref();
```

Increases the reference count by one.

IT_PSS_StorageObjectFactory::create()

```
virtual CosPersistentState::StorageObject* create()
    throw(CORBA::SystemException);
```

Creates and returns a new `StorageObject` object.

Enhancement This is an Orbix enhancement.

**IT_PSS_StorageObjectFactory::
IT_PSS_StorageObjectFactory()**

`IT_PSS_StorageObjectFactory();`

The constructor.

Enhancement This is an Orbix enhancement.

IT_PSS_StorageObjectFactory::_remove_ref()

`virtual void _remove_ref();`

Decreases the reference count by one.

Enhancement This is an Orbix enhancement.

IT_PSS::TransactionalSession Interface

When you create a transactional session with an IONA PSS implementation, you get an `IT_PSS::TransactionalSession` object (the most derived type of this object, however, is `IT_PSS::TransactionalSession2`).

```
// PSDL Code in module IT_PSS
local interface TransactionalSession :
Session, CosPersistentState::TransactionalSession
{
Master get_master();
boolean is_replica();
Replica get_replica();
};
```

This interface provides proprietary enhancements to the OMG `TransactionalSession` interface. It consists of functions to manage replicated persistent objects.

IT_PSS::TransactionalSession::get_master

```
Master get_master();
```

Returns an object reference to a replica's master instance. If the session is associated with a master, then it will return an object reference to itself. If the master instance was not set or is unreachable, the function will return `NIL`.

IT_PSS::TransactionalSession::is_replica

```
boolean is_replica();
```

Returns `TRUE` if the object is a replica of a datastore and `FALSE` if it is not.

IT_PSS::TransactionalSession:get_replica

[Replica](#) get_replica();

If the session is associated with a replica of a datastore, it will return an object reference to its [Replica](#) object. If the session is associated with a master instance, it will return `NIL`.

IT_PSS::TransactionalSession2 Interface

When you create a transactional session with an IONA PSS implementation, you get an `IT_PSS::TransactionalSession2` object, which inherits from the `IT_PSS::TransactionalSession` interface.

```
// PSDL Code in module IT_PSS
local interface TransactionalSession2 :
    TransactionalSession
{
    string
    refresh_master(
        in TimeBase::TimeT timeout
    );
};
```

IT_PSS::TransactionalSession2::refresh_master

```
string refresh_master(in TimeBase::TimeT timeout);
```

Returns the current or new master or "" if there is no current master within a specified timeout.

IT_PSS::TxSessionAssociation Class

You can use stack-allocated TxSessionAssociation objects to create associations between OTS transactions and PSS transactional sessions managed by a SessionManager.

```
class TxSessionAssociation {
public:
    TxSessionAssociation(
        IT_PSS::SessionManager_ptr          session_mgr,
        CosPersistentState::AccessMode      access_mode
    ) throw(CORBA::SystemException);

    TxSessionAssociation(
        IT_PSS::SessionManager_ptr          session_mgr,
        CosPersistentState::AccessMode      access_mode,
        CosTransactions::Coordinator_ptr    tx_coordinator
    ) throw(CORBA::SystemException);

    ~TxSessionAssociation()
        throw(CORBA::SystemException);

    IT_PSS::TransactionalSession_ptr  get_session_nc()
        const throw();

    CosTransactions::Coordinator_ptr  get_tx_coordinator_nc()
        const throw();

    void suspend()
        throw(CORBA::SystemException);

    void end(
        CORBA::Boolean success = IT_TRUE
    ) throw(CORBA::SystemException);

private:
    ...
};
```

Enhancement This class is an Orbix enhancement.

TxSessionAssociation::end()

```
void end(  
    CORBA::Boolean success = IT_TRUE  
) throw(CORBA::SystemException);
```

Ends the association only if this object started or resumed the association. This method has no effect if the association already ended.

Parameters

`success` Determines if the method was successful.

Enhancement This is an Orbix enhancement.

See Also `IT_PSS::TxSessionAssociation::suspend()`

TxSessionAssociation::get_session_nc()

```
IT\_PSS::TransactionalSession_ptr get_session_nc(  
    const IT_THROW_DECL());
```

Returns a non-copied reference to the session. This mean that the caller must not release the returned reference.

Enhancement This is an Orbix enhancement.

TxSessionAssociation::get_tx_coordinator_nc()

```
CosTransactions::Coordinator_ptr get_tx_coordinator_nc(  
    const IT_THROW_DECL());
```

Returns a non-copied reference to the association's transaction coordinator. This mean that the caller must not release the returned reference. After a transaction-session association object is constructed, `get_tx_coordinator_nc()` returns nil when and only when the object represents an association between the session manager's read-only transaction and the session manager's shared read-only session.

Enhancement This is an Orbix enhancement.

TxSessionAssociation::TxSessionAssociation() Constructors

```
TxSessionAssociation(  
    IT\_PSS::SessionManager\_ptr                session_mgr,  
    CosPersistentState::AccessMode            access_mode  
) throw(CORBA::SystemException);
```

A constructor without a supplied transaction.

```
TxSessionAssociation(  
    IT\_PSS::SessionManager\_ptr                session_mgr,  
    CosPersistentState::AccessMode            access_mode,  
    CosTransactions::Coordinator\_ptr         tx_coordinator  
) throw(CORBA::SystemException);
```

A constructor with a transaction.

Parameters

<code>session_mgr</code>	The session manager.
<code>access_mode</code>	Access mode for the association. If <code>tx_coordinator</code> is not provided, the constructor's behavior is as follows: <ul style="list-style-type: none">• If access mode is <code>READ_ONLY</code>, then start or use an association between the session manager's read-only transaction and the session manager's shared read-only session.• If access mode is <code>READ_WRITE</code>, then raise the <code>CORBA::TRANSACTION_REQUIRED</code>.
<code>tx_coordinator</code>	A transaction coordinator.

If a transaction is provided, the behavior depends on the number of associations between this transaction and sessions created by the session's manager connector:

Table 5: *Associations Between a Transaction and Sessions*

Number of Associations	Behavior
Greater than 1	Raises the <code>CORBA::IMPL_LIMIT</code> exception.
1	Does nothing if it is <code>ACTIVE</code> , otherwise it starts it.
none	Creates a new association between this transaction and a read-write transactional session managed by the session manager.

Enhancement This is an Orbix enhancement.

TxSessionAssociation::~~TxSessionAssociation() Destructor

```
~TxSessionAssociation()  
    throw(CORBA::SystemException);
```

If there is still an association when the destructor is called, and this object started the association, the association is suspended. If the suspend fails, the association ends with the success flag set to `FALSE`.

Enhancement This is an Orbix enhancement.

TxSessionAssociation::suspend()

```
void suspend()  
    throw(CORBA::SystemException);
```

Suspends the association only when this object started or resumed the association. This method has no effect if the association has already suspended or ended.

Enhancement This is an Orbix enhancement.

See Also [IT_PSS::TxSessionAssociation::end\(\)](#)

The IT_PSS_DB Module Overview

This module contains the single interface `Env`.

IT_PSS_DB::Env Interface

```
// IDL
module IT_PSS_DB {
    interface Env {
        readonly attribute string name;

        void pre\_backup\(\);
        void post\_backup\(\);
        void checkpoint\(\);
    };
};
```

Enhancement This interface is an Orbix enhancement.

Env::checkpoint()

```
// IDL
void checkpoint();
```

Enhancement This is an Orbix enhancement.

Env::name Attribute

```
// IDL
readonly attribute string name;
```

Enhancement This is an Orbix enhancement.

Env::post_backup()

```
// IDL  
void post_backup();
```

Enhancement This is an Orbix enhancement.

Env::pre_backup()

```
// IDL  
void pre_backup();
```

Enhancement This is an Orbix enhancement.

Index

A

absolute() 72
access_mode attribute 9
AccessMode type 2
_add_ref() 95, 99
after_last() 72
AssociationStatus type 46

B

before_first() 73
block_readers_until_idle() 87

C

cancel_row_updates() 73
_catalog() 40
CatalogBase interface 9, 53
checkpoint() 111
clear_parameters() 63
close() 10, 73, 90
Concurrency Type 73
Connector interface 13, 57
CosPersistentState module 1
CosPersistentState_Factory template class 23
create() 96, 99
create_basic_session() 14
create_transactional_session() 16
current_session() 18

D

default_isolation_level attribute 46
define_parameter() 64
delete_row() 74
destroy_object() 33, 40

E

end() 46, 106
EndOfAssociationCallback interface 25
Env interface 111
execute() 90
execute_prepared() 64
execute_prepared_query() 64
execute_prepared_update() 64
execute_query() 90

execute_update() 91

F

FetchDirection Type 74
find_by_pid() 10
find_by_short_pid() 29
find_state_member() 74
find_storage_home() 10
first() 75
flush() 11
ForUpdate enumeration 2
free_all() 11

G

get() 75
get_association_status() 47
get_by_name() 75
get_catalog() 30, 91
get_concurrency() 76
get_fetch_direction() 76, 91
get_fetch_size() 76, 91
get_master() 101, 103
get_pid() 18, 33, 41
get_replica() 102
get_result_set() 92
get_result_set_concurrency() 92
get_result_set_type() 92
get_row() 76
get_session_nc() 106
get_shared_read_only_session_nc() 87
get_short_pid() 18, 34, 41
get_statement() 77
get_storage_home() 34, 41
get_tx_coordinator_nc() 106
get_type() 77

I

_impl_data() 41
implementation_id attribute 19
insert_row() 77
is_after_last() 77
is_before_first() 77
is_first() 78

Index

is_last() 78
is_null() 41
IsolationLevel type 3
is_replica() 101
it_create_session_manager() 57
it_create_statement() 54
it_create_statement_with_type_and_concurr
ency() 54
it_discard_all() 54
it_discard_flush_list() 55
it_lock() 97
it_prepare_statement() 55
it_prepare_statement_with_type_and_concurr
ency() 55
IT_PSS module 51
IT_PSS_DB module 109
IT_PSS_StorageHomeFactory class 31
IT_PSS_StorageHomeFactory template 95
IT_PSS_StorageHomeFactory() constructor 96
IT_PSS_StorageObjectFactory template 99
IT_PSS_StorageObjectFactory() constructor 100

L

last() 78
last_successful_refresh 68

M

Master interface 61
move_to_current_row() 78
move_to_insert_row() 79

N

name attribute 111
next() 79
NotFound exception 4

O

object_exists() 34
operator=() 41
operator->() 42

P

Parameter structure 4
ParameterList sequence 5
Pid type 5
post_backup() 112
pre_backup() 112
PreparedStatement interface 63

previous() 79

R

refresh() 12, 68
refresh_row() 79
register_session_factory() 19
register_session_pool_factory() 19
register_storage_home_factory() 20
register_storage_object_factory() 20
relative() 80
release() 42
_remove_ref() 96, 100
Replica interface 59, 67
ResultSet interface 69
row_deleted() 80
row_inserted() 80
row_updated() 80

S

same_ref() 42
Session interface 27, 85
SessionManager interface 87
sessions() 20
set() 81
set_by_name() 81
set_fetch_direction() 81, 92
set_fetch_size() 82, 93
set_master() 67
ShortPid type 5
start() 47
Statement interface 89
_static_type() 42
StorageHomeBase interface 29
StorageHomeFactory native type 31
StorageObject interface 33, 97
StorageObjectBase native type 35
StorageObjectFactory native type 37
StorageObjectRef class 39
StorageObjectRef() 42
suspend() 108
suspend() 48

T

_target() 43
_target_type 43
transaction() 49
TransactionalSession interface 45, 101, 103
TransactionalSessionList sequence 5
TxSessionAssociation class 105

TxSessionAssociation() constructors 107

Type 82

TypeId type 6

U

update_row() 83

Y

YieldRef enumeration 6

Index
