Orbix Persistent State Service Reference

V6.3.14

Table of Contents

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.

  WRITER: this is a bunch of random thoughts, hidden from reader. The Persistent State Service uses these terms:

storage object	PSS presents persistent information as storage objects. Each storage object has a type that defines its members and operations.
storage home	Storage objects are stored in storage homes. A storage home can only contain storage objects of a given type. It also defines operations and keys. A storage home can be a base from which other storage homes are derived, and it manages its own storage objects along with those objects of its derived storage homes.
datastore	A datastore is an entity that manages data, for example a database, a set of files, or a schema in a relational database. Storage homes are stored in datastores where there is at most one storage home for a given data type. A storage object instance may be bound to a storage object in the datastore, and provides direct access to the state of this storage object. Updating the instance updates the storage object in the datastore.
catalog	Catalogs manage storage homes and provide instances of storage homes. A session is a kind of catalog.

session	To access a storage object, your application needs a logical connection, called a *session*, between your process and the datastore that contains the storage home of this storage object. Sessions give you programmatic control over session-allocation and session-transaction association. A session can give access to more than one datastore. Session management is either explicit (you create and manage sessions for your application) or implicit (you create one or more session pools that manage sessions for you). Sessions and session pools are the two kinds of catalogs of PSS. Sessions or a session pool is created by `Connector` functions.

Access to storage objects can be either transactional or non-transactional. The programming model with transactions is slightly different than it is without transactions. With transactions, the application must start and end transactions.

Your use the *Persistent State Definition Language* (PSDL) to define storage objects and storage homes for your applications. PSDL is a superset of the OMG's IDL and is a declarative language (like IDL) not a programming language. For example, storage objects can have state members and operations parameters of any IDL type. PSDL obeys the same lexical rules as IDL except that it has some new keywords. Its grammar is an extended IDL grammar, with new constructs to define storage objects and storage homes. A PSDL specification can contain any IDL construct; further, local operations (on local interface, values, storage objects, storage homes and catalogs) can accept parameters of PSDL types, such as a sequence of storage object references. A source file containing PSDL constructs must has a `.psdl` extension. The file `CosPersistentState.psdl` contains PSDL type definitions and is implicitly included in any PSDL specification.

The `CosPersistentState` module's features are listed in Table 1:

Table 1 The CosPersistentState Module

Common Data Types	Interfaces
Type Enumeration Type Exception Structure Sequence Type Sequence Type Sequence Type Type Enumeration	SessionPool
	Native Types and Helper Classes

	SessionFactory SessionPoolFactory

```
// PSDL From File: CosPersistentState.psdl. (Jan 2000, about Beta2)
```

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.

ExamplesFor 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:

`READ_UNCOMMITTED`	When a resource has this isolation level, its user may experience the dirty reads and the non-repeatable reads phenomena.
`READ_COMMITTED`	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::PidList Sequence

```
NOT USED FOR GA
// PSDL Code
   typedef sequence<Pid> PidList;
```

A list of process identifiers.

See Also CosPersistentState::Pid

```
CosPersistentState::SessionPool
```

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::TransactionPolicy Type

NOT IMPLEMENTED FOR GA - (SessionPool stuff)

```
// PSDL Code
    typedef short TransactionPolicy;
    const TransactionPolicy NON_TRANSACTIONAL = 0;
    const TransactionPolicy TRANSACTIONAL     = 1;
```

Valid values include:

```
NON_TRANSACTIONAL
TRANSACTIONAL
```

See Also CosPersistentState::Connector::create_session_pool()

```
CosPersistentState::SessionPool::transaction_policy()
```

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 AlsoCORBA::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.

ExamplesFor 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_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.

```
NOTE: C++ code From post-Beta2 January 2000
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 From File: CosPersistentState.psdl. (Jan 2000, about Beta2)
// PSDL code in module CosPersistentState
local interface EndOfAssociationCallback {
    void released(in TransactionalSession session);
};
```

See Also CosPersistentState::Connector::create_transactional_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 From File: CosPersistentState.psdl. (Jan 2000, about Beta2)
// PSDL in module CosPersistentState
local interface StorageHomeBase {
    StorageObjectBase [find_by_short_pid](#cospersistentstatestoragehomebase-
interface)(
        in ShortPid short_pid
    )
    raises (NotFound);
    CatalogBase [get_catalog()](#it_pssstatement-interface);
};
```

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

```
CosPersistentState::NotFound
```

The object is not found.

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 From File: CosPersistentState.psdl. (Jan 2000, about Beta2)
// PSDL in module CosPersistentState
native StorageHomeFactory;
```

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

```
// In file omg/CosPersistentState_natives.h
// C++
typedef CosPersistentState_Factory<StorageHomeBase> StorageHomeFactory;
```

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

See AlsoIT_PSS_StorageHomeFactory

```
CosPersistentState::CosPersistentState_Factory
```

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 From File: CosPersistentState.psdl. (Jan 2000, about Beta2)
// PSDL in module CosPersistentState
    native StorageObjectBase;
WRITER NOTE: C++ Code from post Beta2, January 2000
```

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

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

CosPersistentState::StorageObjectFactory Native Type

`StorageObjectFactory` is a native type.

```
// PSDL From File: CosPersistentState.psdl. (Jan 2000, about Beta2)
// in module CosPersistentState
native StorageObjectFactory;
```

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

```
// In file omg/CosPersistentState_natives.h
// C++
typedef CosPersistentState_Factory<StorageObject> StorageObjectFactory;
```

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

See AlsoIT_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](#cospersistentstatestorageobjectref-
class);
    static CORBA::TypeCode_ptr [_static_type()]
(#cospersistentstatestorageobjectref-class);
    [StorageObjectRef](#cospersistentstatestorageobjectref-class)(
        StorageObject*  obj       = 0,
        CatalogBase_ptr catalog   = 0,
        void*           impl_data = 0
    );
    [StorageObjectRef](#cospersistentstatestorageobjectref-class)(
        const StorageObjectRef& ref
    );
    StorageObjectRef&  [operator=](#cospersistentstatestorageobjectref-class)
(
        const StorageObjectRef& ref
    );
    StorageObjectRef&  [operator=](#cospersistentstatestorageobjectref-class)
(
        StorageObject* obj
    );
    void [release()](#cospersistentstatestorageobjectref-class);
    StorageObject* [operator->()](#cospersistentstatestorageobjectref-
class); // not const!
    CORBA::Boolean [same_ref](#cospersistentstatestorageobjectref-class)(
        StorageObjectRef
    ) const;
    void [destroy_object()](#cospersistentstatestorageobjectref-class) const;
    Pid* [get_pid()](#cospersistentstatestorageobjectref-class) const;
    ShortPid* [get_short_pid()](#cospersistentstatestorageobjectref-class)
const;
    CORBA::Boolean [is_null()](#cospersistentstatestorageobjectref-class)
const;
    StorageHomeBase_ptr [get_storage_home()]
(#cospersistentstatestorageobjectref-class) const;
    // read-only access to data members
    void* [_impl_data()](#cospersistentstatestorageobjectref-class) const;
    CosPersistentState::CatalogBase_ptr [_catalog()]
(#cospersistentstatestorageobjectref-class) const;
    StorageObject* [_target()](#cospersistentstatestorageobjectref-class)
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.

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 from January 2000, post Beta2
// PSDL Code in Module IT_PSS
local interface CatalogBase :
CosPersistentState::CatalogBase {
    Statement [it_create_statement()](#it_psscatalogbase-interface);
    Statement [it_create_statement_with_type_and_concurrency]
(#it_psscatalogbase-interface)(
        in ResultSet::Type type,
        in ResultSet::Concurrency concurrency
    );
    PreparedStatement [it_prepare_statement](#it_psscatalogbase-interface)(
        in string pssql
    );
    PreparedStatement
    [it_prepare_statement_with_type_and_concurrency](#it_psscatalogbase-
interface)(
        in string pssql,
        in ResultSet::Type type,
        in ResultSet::Concurrency concurrency
    );
    void [it_discard_flush_list()](#it_psscatalogbase-interface);
    void [it_discard_all](#it_psscatalogbase-interface)(
        in boolean clear_non_id_refs
    );
};
```

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis is an Orbix enhancement.

CatalogBase::it_discard_flush_list()

// PSDL Code

```
void it_discard_flush_list();
```

Discards all modified objects in the catalog.

EnhancementThis 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.

EnhancementThis 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 `ResutSet` type. The concurrency setting can be either read-only or updateable.

EnhancementThis is an Orbix enhancement.

IT_PSS::Connector Interface

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

```
// PSDL Code from January 2000, post Beta2
// PSDL Code in module IT_PSS
/* local */ interface Connector : CosPersistentState::Connector {
    SessionManager [it_create_session_manager](#it_pssconnector-interface)(
        in CosPersistentState::ParameterList parameters
    );
};
```

EnhancementThis 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 1 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 1 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.

EnhancementThis is an Orbix enhancement.

See AlsoIT_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 from January 2000, post Beta2
// PSDL Code in module IT_PSS
local interface PreparedStatement : Statement {
    void [execute_prepared()](#it_psspreparedstatement-interface);
    ResultSet [execute_prepared_query()](#it_psspreparedstatement-interface);
    unsigned long [execute_prepared_update()](#it_psspreparedstatement-
interface);
    void [define_parameter](#it_psspreparedstatement-interface)(
        in unsigned short parameter_index,
        in any parameter_value
    );
    void [clear_parameters()](#it_psspreparedstatement-interface);
};
```

EnhancementThis is an Orbix enhancement.

See AlsoIT_PSS::CatalogBase

```
IT_PSS::Statement
```

PreparedStatement::clear_parameters()

// PSDL Code

```
void clear_parameters();
```

Clears the current parameter values immediately.

EnhancementThis 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.

EnhancementThis is an Orbix enhancement.

PreparedStatement::execute_prepared()

// PSDL Code

```
void execute_prepared();
```

Executes the prepared SQL statement.

EnhancementThis is an Orbix enhancement.

See AlsoIT_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.

EnhancementThis is an Orbix enhancement.

See AlsoIT_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.

EnhancementThis is an Orbix enhancement.

See AlsoIT_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 replcia. 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.

ParametersThis 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](#it_pssresultset-interface) Type
[FetchDirection](#it_pssresultset-interface) Type
[Type](#it_pssresultset-interface)
```

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()`

EnhancementThis interface is an Orbix enhancement.

See AlsoIT_PSS::CatalogBase

```
// PSDL Code from January 2000, post Beta2
// PSDL Code in module IT_PSS
local interface ResultSet {
    typedef unsigned short [Type](#it_pssresultset-interface);
    const Type TYPE_FORWARD_ONLY =        1;
    const Type TYPE_SCROLL_INSENSITIVE =  2;
    const Type TYPE_SCROLL_SENSITIVE =    3;
    typedef unsigned short [Concurrency](#it_pssresultset-interface);
    const Concurrency CONCUR_READ_ONLY = 1;
    const Concurrency CONCUR_UPDATABLE = 2;
    typedef unsigned short [FetchDirection](#it_pssresultset-interface);
    const FetchDirection FETCH_FORWARD = 1;
    const FetchDirection FETCH_REVERSE = 2;
    const FetchDirection FETCH_UNKNOWN = 3;
    Statement [get_statement()](#it_pssresultset-interface);
    // Basic operations
    //
    boolean [next()](#it_pssresultset-interface);
    void [close()](#it_pssstatement-interface);
    any [get](#it_pssresultset-interface)(
        in unsigned short index
    );
    any [get_by_name](#it_pssresultset-interface)(
        in string state_member_name
    );
    // Find state_member
    //
    unsigned short [find_state_member](#it_pssresultset-interface)(
       in string state_member_name
    );
    // Getting/setting the current row
    //
    boolean [is_after_last()](#it_pssresultset-interface);
    boolean [is_before_first()](#it_pssresultset-interface);
    boolean [is_first()](#it_pssresultset-interface);
    boolean [is_last()](#it_pssresultset-interface);
    void [after_last()](#it_pssresultset-interface);
    void [before_first()](#it_pssresultset-interface);
    boolean [first()](#it_pssresultset-interface);
    boolean [last()](#it_pssresultset-interface);
    unsigned short [get_row()](#it_pssresultset-interface);
    boolean [absolute](#it_pssresultset-interface)(
        in short row
    );
    boolean [relative](#it_pssresultset-interface)(
        in short rows
```

```
    );
    boolean [previous()](#it_pssresultset-interface);
    void [move_to_insert_row()](#it_pssresultset-interface);
    void [move_to_current_row()](#it_pssresultset-interface);
    // Fetch direction and size
    //
    void [set_fetch_direction](#it_pssstatement-interface)(
        in FetchDirection direction
    );
    FetchDirection [get_fetch_direction()](#it_pssstatement-interface);
    void [set_fetch_size](#it_pssstatement-interface)(
        in unsigned short fetch_size
    );
    unsigned short [get_fetch_size()](#it_pssstatement-interface);
    // Type and Concurrency
    //
    Type [get_type()](#it_pssresultset-interface);
    Concurrency [get_concurrency()](#it_pssresultset-interface);
    // Was row modified?
    //
    boolean ;
    boolean [row_inserted()](#it_pssresultset-interface);
    boolean [row_deleted()](#it_pssresultset-interface);
    // Write operations
    //
    void [set](#it_pssresultset-interface)(
        in unsigned short index,
        in any value
    );
    void [set_by_name](#it_pssresultset-interface)(
        in string state_member_name,
        in any value
    );
    void [insert_row()](#it_pssresultset-interface);
    void [update_row()](#it_pssresultset-interface);
    void [delete_row()](#it_pssresultset-interface);
    void [refresh_row()](#it_pssresultset-interface);
    void [cancel_row_updates()](#it_pssresultset-interface);
};
```

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.

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis is an Orbix enhancement.

ResultSet::cancel_row_updates()

// PSDL Code

```
void cancel_row_updates();
```

Cancels the updates made to a row in the table.

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis is an Orbix enhancement.

ResultSet::delete_row()

// PSDL Code

```
void delete_row();
```

Deletes the current row from the table.

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis 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

EnhancementThis is an Orbix enhancement.

ResultSet::get()

// PSDL Code

```
any get(
    in unsigned short index
);
```

Returns the value for the given parameter index.

EnhancementThis is an Orbix enhancement.

See AlsoIT_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.

EnhancementThis is an Orbix enhancement.

See AlsoIT_PSS::ResultSet::set_by_name()

ResultSet::get_concurrency()

// PSDL Code

```
Concurrency get_concurrency();
```

Returns the concurrency value.

EnhancementThis is an Orbix enhancement.

ResultSet::get_fetch_direction()

// PSDL Code

```
FetchDirection get_fetch_direction();
```

Returns the direction of table row processing.

EnhancementThis is an Orbix enhancement.

See AlsoIT_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.

EnhancementThis is an Orbix enhancement.

See AlsoIT_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.

EnhancementThis is an Orbix enhancement.

ResultSet::get_statement()

// PSDL Code

```
Statement get_statement();
```

Returns the Statement that produced this `ResultSet` object.

EnhancementThis 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.

EnhancementThis is an Orbix enhancement.

ResultSet::insert_row()

// PSDL Code

```
void insert_row();
```

Inserts a row.

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis is an Orbix enhancement.

ResultSet::move_to_insert_row()

// PSDL Code

```
void move_to_insert_row();
```

Moves the cursor to the insert row.

EnhancementThis 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.

EnhancementThis is an Orbix enhancement.

ResultSet::previous()

// PSDL Code

```
boolean previous();
```

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

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis is an Orbix enhancement.

ResultSet::set()

// PSDL Code

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

Sets the value and parameter index.

EnhancementThis is an Orbix enhancement.

See AlsoIT_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.

EnhancementThis is an Orbix enhancement.

See AlsoIT_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.

EnhancementThis is an Orbix enhancement.

See AlsoIT_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.

EnhancementThis is an Orbix enhancement.

See AlsoIT_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.

EnhancementThis 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.

EnhancementThis 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 from January 2000, post Beta2
// PSDL Code in module IT_PSS
local interface Session : CatalogBase, CosPersistentState::Session
    {};
```

EnhancementThis interface is an Orbix enhancement.

See AlsoIT_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, and a pool of read-write serializable transactional sessions for write requests and for request executed in the context of a distributed transaction. Of course, you can also create and manage your transactional sessions directly with the standard lower level PSS APIs from the `CosPersistentState` module.

```
// PSDL Code from January 2000, post Beta2
//PSDL in module IT_PSS
local interface SessionManager {
    TransactionalSession [get_shared_read_only_session_nc()]
(#it_psssessionmanager-interface);
    void [block_readers_until_idle()](#it_psssessionmanager-interface);
};
```

EnhancementThis interface is an Orbix enhancement.

See AlsoIT_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.

EnhancementThis 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.

EnhancementThis 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 from January 2000, post Beta2
// PSDL Code in module IT_PSS
local interface Statement {
    void [execute](#it_pssstatement-interface)(
        in string pssql
    );
    ResultSet [execute_query](#it_pssstatement-interface)(
        in string pssql
    );
    unsigned long [execute_update](#it_pssstatement-interface)(
        in string pssql
    );
    ResultSet [get_result_set()](#it_pssstatement-interface);
    void [close()](#it_pssstatement-interface);
    // Default fetch direction and size
    //
    void [set_fetch_direction](#it_pssstatement-interface)(
        in ResultSet::FetchDirection direction
    );
    ResultSet::FetchDirection [get_fetch_direction()](#it_pssstatement-
interface);
    void [set_fetch_size](#it_pssstatement-interface)(
        in unsigned short fetch_size
    );
    unsigned short [get_fetch_size()](#it_pssstatement-interface);
    // Type and Concurrency
    //
    ResultSet::Type [get_result_set_type()](#it_pssstatement-interface);
    ResultSet::Concurrency [get_result_set_concurrency()](#it_pssstatement-
interface);
    CatalogBase [get_catalog()](#it_pssstatement-interface);
};
```

EnhancementThis interface is an Orbix enhancement.

See AlsoIT_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.

EnhancementThis is an Orbix enhancement.

Statement::execute()

// PSDL Code

```
void execute(
    in string pssql
);
```

Executes an SQL statement that may obtain multiple results.

EnhancementThis is an Orbix enhancement.

Statement::execute_query()

// PSDL Code

```
ResultSet execute_query(
    in string pssql
);
```

Executes an SQL statement that returns a single ResultSet.

EnhancementThis is an Orbix enhancement.

Statement::execute_update()

// PSDL Code

```
unsigned long execute_update(
    in string pssql
);
```

Executes an SQL `INSERT`, `UPDATE` or `DELETE` statement.

EnhancementThis is an Orbix enhancement.

Statement::get_catalog()

// PSDL Code

```
CatalogBase get_catalog();
```

Returns the catalog for this Statement.

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis is an Orbix enhancement.

Statement::get_result_set()

// PSDL Code

```
ResultSet get_result_set();
```

Returns the current result as a ResultSet object.

EnhancementThis is an Orbix enhancement.

Statement::get_result_set_concurrency()

// PSDL Code

```
ResultSet::Concurrency get_result_set_concurrency();
```

Returns the result set concurrency.

EnhancementThis is an Orbix enhancement.

Statement::get_result_set_type()

// PSDL Code

```
ResultSet::Type get_result_set_type();
```

Returns the type of the ResultSet.

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis is an Orbix enhancement.

IT_PSS_StorageHomeFactory Template

Use this template class to help implement your StorageHomeFactory.

```
// In file orbix_sys/pss_xtra.h (Jan 2000, about Beta2)
template<class T>
class IT_PSS_StorageHomeFactory :
public CosPersistentState::StorageHomeFactory {
  public:
    [IT_PSS_StorageHomeFactory()](#it_pss_storagehomefactory-template);
    virtual void [_add_ref()](#it_pss_storageobjectfactory-template);
    virtual void [_remove_ref()](#it_pss_storageobjectfactory-template);
    virtual CosPersistentState::StorageHomeBase_ptr [create()]
(#it_pss_storageobjectfactory-template)
        throw(CORBA::SystemException);
  private:
    ...
};
```

EnhancementThis is an Orbix enhancement.

IT_PSS_StorageHomeFactory::_add_ref()

virtual void _add_ref();

Increases the reference count by one.

EnhancementThis 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.

EnhancementThis is an Orbix enhancement.

IT_PSS_StorageHomeFactory::_remove_ref()

virtual void _remove_ref();

Decreases the reference count by one.

EnhancementThis 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`.

```
Writer: I know this needs some real description.
// PSDL Code from January 2000, post Beta2
// PSDL Code in module IT_PSS
abstract storagetype StorageObject {
    void [it_lock()](#it_pssstorageobject-interface);
    // boolean it_is_locked() const;
    };
```

EnhancementThis 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.

EnhancementThis is an Orbix enhancement.

IT_PSS_StorageObjectFactory Template

Use this template class to help implement your StorageObjectFactory.

```
// In file orbix_sys/pss_xtra.h (Jan 2000, about Beta2)
// c++
template<class T>
class IT_PSS_StorageObjectFactory :
public CosPersistentState::StorageObjectFactory {
  public:
    [IT_PSS_StorageObjectFactory()](#it_pss_storageobjectfactory-template);
    virtual void  [_add_ref()](#it_pss_storageobjectfactory-template);
    virtual void  [_remove_ref()](#it_pss_storageobjectfactory-template);
    virtual CosPersistentState::StorageObject* [create()]
(#it_pss_storageobjectfactory-template)
        throw(CORBA::SystemException);
  private:
    ...
};
```

EnhancementThis 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.

EnhancementThis is an Orbix enhancement.

IT_PSS_StorageObjectFactory::IT_PSS_StorageObjectFactory()

IT_PSS_StorageObjectFactory();

The constructor.

EnhancementThis is an Orbix enhancement.

IT_PSS_StorageObjectFactory::_remove_ref()

virtual void _remove_ref();

Decreases the reference count by one.

EnhancementThis 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 propriatary 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_psstxsessionassociation-class)(
        IT_PSS::SessionManager_ptr              session_mgr,
        CosPersistentState::AccessMode          access_mode
    ) throw(CORBA::SystemException);
    [TxSessionAssociation](#it_psstxsessionassociation-class)(
        IT_PSS::SessionManager_ptr              session_mgr,
        CosPersistentState::AccessMode          access_mode,
        CosTransactions::Coordinator_ptr        tx_coordinator
    ) throw(CORBA::SystemException);
    [~TxSessionAssociation()](#it_psstxsessionassociation-class)
        throw(CORBA::SystemException);
    IT_PSS::TransactionalSession_ptr  [get_session_nc()]
(#it_psstxsessionassociation-class)
        const throw();
    CosTransactions::Coordinator_ptr  [get_tx_coordinator_nc()]
(#it_psstxsessionassociation-class)
        const throw();
    void [suspend()](#it_psstxsessionassociation-class)
        throw(CORBA::SystemException);
    void [end](#it_psstxsessionassociation-class)(
        CORBA::Boolean success = IT_TRUE
    ) throw(CORBA::SystemException);
  private:
    ...
};
```

EnhancementThis 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.

EnhancementThis is an Orbix enhancement.

See Also 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.

EnhancementThis 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.

EnhancementThis 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

`session_mgr`	The session manager.
`access_mode`	Access mode for the association. If `tx_coordinator` is not provided, the constructor's behavior is as follows: - If access mode is `READ_ONLY`, 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 `READ_WRITE`, then raise the `CORBA::TRANSACTION_REQUIRED`.
`tx_coordinator`	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 1 Associations Between a Transaction and Sessions

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

EnhancementThis 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.

EnhancementThis 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.

EnhancementThis is an Orbix enhancement.

See AlsoIT_PSS::TxSessionAssociation::end()

The IT_PSS_DB Module Overview

This module contains the single interface `Env`.

IT_PSS_DB::Env Interface

```
// IDL From January 2000
// IDL
module IT_PSS_DB {
    interface Env {
        readonly attribute string [name](#it_pss_dbenv-interface);
        void [pre_backup()](#it_pss_dbenv-interface);
        void [post_backup()](#it_pss_dbenv-interface);
        void [checkpoint()](#it_pss_dbenv-interface);
    };
};
```

EnhancementThis interface is an Orbix enhancement.

Env::checkpoint()

// IDL

```
void checkpoint();
```

EnhancementThis is an Orbix enhancement.

Env::name Attribute

// IDL

```
readonly attribute string name;
```

EnhancementThis is an Orbix enhancement.

Env::post_backup()

// IDL

```
void post_backup();
```

EnhancementThis is an Orbix enhancement.

Env::pre_backup()

// IDL

```
void pre_backup();
```

EnhancementThis is an Orbix enhancement.

Notices

Copyright

Trademarks

Examples

This information might contain examples of data and reports. The examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

License agreement

Corporate information

Rocket Software, Inc. develops enterprise infrastructure products in four key areas: storage, networks, and compliance; database servers and tools; business information and analytics; and application development, integration, and modernization.

Website: www.rocketsoftware.com

Contacting Technical Support

The Rocket Community is the primary method of obtaining support. If you have current support and maintenance agreements with Rocket Software, you can access the Rocket Community and report a problem, download an update, or read answers to FAQs. To log in to the Rocket Community or to request a Rocket Community account, go to www.rocketsoftware.com/support. In addition to using the Rocket Community to obtain support, you can use one of the telephone numbers that are listed above or send an email to support@rocketsoftware.com.

Rocket Global Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451-1468
USA

Country and Toll-free telephone number

To contact Rocket Software by telephone for any reason, including obtaining pre-sales information and technical support, use one of the following telephone numbers.

- United States: 1-855-577-4323
- Australia: 1-800-823-405
- Belgium: 0800-266-65
- Canada: 1-855-577-4323
- China: 400-120-9242
- France: 08-05-08-05-62
- Germany: 0800-180-0882
- Italy: 800-878-295
- Japan: 0800-170-5464
- Netherlands: 0-800-022-2961
- New Zealand: 0800-003210
- South Africa: 0-800-980-818
- United Kingdom: 0800-520-0439