Glossary

abort

An abnormal termination of a transaction. When a transaction aborts, its changes to the database are erased, and the database is effectively restored to its state as of the moment the transaction began. See the function **os_transaction::abort()** in Using Transactions, Chapter 3, Transactions, in *ObjectStore Basic API User Guide*.

See also *commit*, *commseg*, *deadlock*, *locking*, *template instantiation*, *transaction*. Also see **os_transaction::abort()** in the *ObjectStore C++ API Reference*.

address space

A range of virtual memory addresses. For most 32-bit computers, the address space is slightly less than 2^{32} bytes. (The platform's virtual memory system might reserve or might not implement a portion of the 2^{32} range.) For 64-bit computers, the address space is substantially larger.

application schema

A schema associated with each ObjectStore application, consisting of all the classes whose instances the application allocates in persistent memory or uses as entry points, as well as all types reachable from these types by navigation.

See also *application schema database*, *compilation schema database*, *database schema*.

application schema object file and **application schema source file**

This file records the location of the application schema database along with the names of the application's virtual function dispatch tables, the names of discriminant functions, and the definitions for any **get_os_typespec()** member functions. Distinctions between the schema object and source files are discussed in Chapter 3, Generating Schemas, of *ObjectStore Building Applications*.

application schema database

An ObjectStore database into which an ObjectStore application's schema is captured when you compile and link the application.

See also *compilation schema database*, *database schema*.

associative access

The selection of those data structures whose field values satisfy a certain condition, such as lookup of an object by name or ID number.

See also *index*, *navigational access*, *query processing*.

atomic

Discrete, committed transactions.

bag

An unordered collection, sometimes called a multiset. Unlike sets, objects can occur in a bag more than once at a given time. Each bag is an instance of the class **os_Bag** or **os_bag**, a class in the ObjectStore Class Library and part of the ObjectStore collection facility.

See also *class extent*, *collection*, *collection cardinality*, *cursor*, *element*, *endian types*, *iteration*, *list*, *multivalued*, *query processing*, *set*. Also see **os_Bag** and **os_bag** in the *ObjectStore Collections API Reference*.

binary relationship

A relationship between two objects, such as that between a part and one of its subparts. Inverses can be used to ensure the integrity of data representing binary relationships.

See also *integrity control*, *inverse*.

Cache Manager

An ObjectStore process that facilitates concurrent access to ObjectStore data by handling callback messages from the Server to client applications. Note that the Cache Manager never reads the cache itself, but coordinates access by clients to cached data.

See also *client cache*.

cache replacement

The process of evicting a page from the client cache to make room for newly retrieved data. In most cases, ObjectStore makes room by evicting some of the least recently used pages.

See also *client cache*, *eviction*, *wired page*.

class

The definition of the data used to represent an object, and the operations that can be performed upon that data.

class extent

A collection containing pointers or ObjectStore references to all the instances of a class.

See also *collection*, *ObjectStore reference*.

client cache

A holding area in the memory of an ObjectStore client, used for data mapped or waiting to be mapped into virtual memory.

See also *Cache Manager*.

clustering

Controlling the arrangement of objects in database memory for the purpose of enhancing locality of reference. Disk and network transfers can be minimized by clustering those objects that are referenced together in programs, .

See also *object faulting*, *persistence*, *segment*.

collection

An object, such as a set, bag, or list, derived from **os_collection**, that serves to group together other objects. They are used to model one-to-many and many-to-many relationships, and they provide a domain for iteration and for the execution of queries.

See also *bag*, *collection cardinality*, *class extent*, *cursor*, *element*, *endian types*, *iteration*, *list*, *multivalued*, *query processing*, *set*.

collection cardinality

The number of elements of a collection, or the sum of the number of occurrences of each element for collections that allow duplicates. For collections like sets, for which each element can occur only once, the cardinality of the collection is simply the number of elements it has. But for collections like bags, in which elements can occur more than once, the number of occurrences of each element must be summed to obtain the cardinality.

See also *bag*, *collection*, *element*, *list*, *multivalued*, *set*. Also see **os_collection::cardinality()** in the *ObjectStore Collections API Reference*.

commit

The normal termination of a transaction. When a nonnested transaction commits, its changes to persistent memory are made permanent and visible. In addition, all its locks on pages read or written are released.

See also *abort*, *commseg*, *deadlock*, *locking*, *transaction*. Also see **os_transaction::commit()** in the *ObjectStore C++ API Reference*.

commseg

Communications segment. This is where the Cache Manager maintains information about permissions on pages and whether or not the client is actually using the page. Each client has its own commseg. For every page in the cache, there is a corresponding item in the commseg.

compilation schema database

This database contains all of an application's schema except the classes used by libraries with which the application links.

See also *application schema database*, *database schema*.

complex path

An index path involving multiple member accesses.

See also *index path*. Also see **os_index_path** in the *ObjectStore Collections API Reference*.

compactor

The compactordefragments the deleted space in an ObjectStore database. See Chapter 6, Compaction, in the *ObjectStore Advanced API User Guide* for details.

concurrency control

A mechanism to provide multiuser access to persistent data in a logically consistent manner.

See also *abort*, *commit*, *locking*, *transaction*.

cursor

An object used to iterate over the elements of a collection.

See also *bag*, *collection*, *iteration*, *index path*, *set*. Also see **os_Cursor** and **os_cursor** in the *ObjectStore Collections API Reference*.

compilation schema or **compilation schema database**

A compilation schema contains information about the application's persistent types, but does not contain information about any persistent types used by any libraries that the application links with.

database

Persistent storage is organized into databases.

See also *commseg*, *database entry point*, *database root*, *database schema*, *database segment*, *persistence*. Also see **os_database_root** in the *ObjectStore C++ API Reference*.

database entry point

A persistent object that serves as a starting point for the retrieval of other persistent objects by means of navigation or query. An entry point is retrieved by means of either a persistent variable or database root.

See also *database root*, *navigational access*, *query processing*.

database root

An instance of the system-supplied class **os_database_root**, it serves to associate a name with a database entry point. Once the association is made, it is possible to retrieve a pointer to the entry point by performing a lookup on the name.

See also *database entry point*, *navigational access*, *query processing*.

database schema A schema associated with an ObjectStore database. Each database has exactly one database schema associated with it. The database schema includes all C++ types of objects that have ever been stored in that database.

database segment Each database is made up of segments, which are used to control the clustering of persistent data. They are variable-sized regions of memory that can be used as the unit of transfer from persistent storage to program memory.

See also *clustering*, *database*, *persistence*, *schema segment*. Also see **os_segment** in the *ObjectStore C++ API Reference*.

data member Class data members declare the objects in the class.

See *class*, *member function*.

deadlock A simple deadlock occurs when one transaction holds a lock on a data item that another transaction is waiting to access, while at the same time the second transaction holds an exclusive lock on a data item that the first transaction is waiting to access. Neither process can proceed until the other does.

See also *abort*, *commseg*, *locking*, *transaction*. Also see Appendix A, Exception Facility, in the *ObjectStore C++ API Reference*.

delete operator Can be used to reclaim both persistent and transient storage.

See also *persistence*, *persistent new*.

discriminant function For each union type with persistent values, a user-provided function that indicates the field of the union currently in use.

See also *persistence*, *union*.

element A value grouped together with other objects in a collection.

See also *bag*, *collection*, *endian types*, *list*, *set*.

element type The type of element a given collection contains. Element types are always pointer types.

See also *collection*, *element*.

endian types Specifies whether the high-order byte is first or the low-order byte is first. A big-endian type specifies the high-order byte first; little-endian types specify the opposite. This information is important in building heterogeneous applications. See Endian Types for

ObjectStore Platforms in Chapter 5, Building Applications for Use on Multiple Platforms, of *ObjectStore Building Applications*.

entry point

See *database entry point*.

eviction

The process of effectively removing a page from the client cache. This process consists of unmapping the page from memory, as well as possibly overwriting the page with newly retrieved data, and possibly transferring the page's data from the client cache to the database.

See also *cache replacement*, *client cache*.

exception facility

This facility provides a mechanism for managing error handling in a C++ application. The facility also enables users to define new exceptions, catch signaled exceptions and redirect flow of control to a handler for the exception, write code that can signal predefined or user-defined exceptions, and specify undo processing to be performed whenever the signaling of an exception redirects control out of or through a given stack frame.

See also *objectstore_exceptionclass*, *tix_exception class*.

file database

An ObjectStore database stored as an operating system file, as opposed to a rawfs database.

See also *database*, *rawfs database*.

heterogeneous

An application that is on multiple platforms and then used to store and update data interchangeably on any of these platforms.

IDE

Visual C++ Integrated Development Environment.

illegal pointer

A pointer stored in a database, the dereferencing of which can result in access to arbitrary memory. ObjectStore can detect, if desired, illegal cross-database pointers and illegal pointers to transient memory.

See also *schema evolution*.

index

A data structure used to provide fast, associative retrieval of collection elements based on a given key, specified with an index path. There are several kinds of indexes, including ordered (implemented as a B-tree) and unordered (implemented as a hash table).

See also *collection*, *iteration*, *index path*, *query optimizer*, *query processing*.

indexable data member

Each data member marked as indexable provides support for automatic maintenance of indexes keyed by the path.

See also *index*, *iteration*.

index path

Paths are used in specifying iteration order, as well as in specifying index keys to enable query optimization. Each path determines a certain kind of mapping by specifying, roughly, a sequence of member names.

See also *index*, *iteration*, *query optimizer*.

instance

An object of a given type is said to be an instance of that type.

instance migration

The phase of schema evolution during which instances of modified classes are altered to conform to new class definitions.

See also *schema evolution*.

integrity control

Compile-time checking and facilities that guarantee the integrity of data that models binary relationships and that detect illegal pointers, such as pointers from persistent to transient memory.

See also *binary relationship*, *inverse*.

inverse

When two data members are specified as inverses of one another, they can be used to form bidirectional links. An update to one data member will automatically trigger a corresponding update to the other member, maintaining consistency between the members. One or both members can be multivalued, or both can be single-valued.

See also *integrity control*, *multivalued*.

iteration

The process of retrieving the elements of a collection one at a time.

See also *cursor*, *collection*.

libraries

Libraries allow multiple programs to share code without recompiling the source. Libraries are specified at link time.

See also *collection*, *recursive query*, *library schema or library schema database*.

library schema or **library schema database**

A library schema contains definitions of persistently allocated types that users of the library do not have access to. If your application uses a library that stores or retrieves persistent data, use the schema generator to create a library schema for that library.

See also *application schema*, *compilation schema database*, *libraries*.

list

An ordered collection in which elements can occur more than once. Each list is an instance of the class **os_List** or **os_list**, a class in the ObjectStore Class Library and part of the ObjectStore collection facility. The class's member functions support various forms of manipulation of lists, such as element insertion and removal.

See also *bag*, *collection*, *collection cardinality*, *class extent*, *cursor*, *element*, *iteration*, *multivalued*, *query processing*, *set*. Also see **os_List** or **os_list** in the *ObjectStore Collections API Reference*.

locking

The mechanism for restricting a process's access to data in order to ensure data integrity and serialization while the data is in use by another process.

See also *abort*, *commit*, *deadlock*, *transaction*.

log file

Each Server keeps a *transaction log*, also called a log file. The most important function of the transaction log is to prevent database corruption in case of failure. The log contains modified pages of data and records about modified pages of data.

mangling

The transformation of C++ identifiers by the C++ compiler into names acceptable to the linker in order to guarantee the identifiers' uniqueness.

mapaside

A technique for ensuring the consistency of data pages in multithreaded applications.

member function

Class member functions declare the operations that can be performed on data members.

See also *class*, *data member*.

MFC

Microsoft Foundation Classes — Visual C++ provides this general purpose class library.

migration

See *instance migration*.

multivalued

Some data members that are, strictly speaking, collection-valued can be viewed as multivalued, where the elements of the collection are viewed as the various values of the data member. Collection-valued data members with inverses are usually treated as multivalued for the purpose of enforcing the inverse constraint.

See also *inverse*, *collection*.

mutex lock

The data structure that coordinates threads. This data structure is also referred to as the global mutex because one mutex coordinates the serialization of all threads.

navigational access

The access of data by following pointers contained in data structure fields. In C++, the data member access syntax supports navigational data access.

See also *associative access*, *query processing*.

neutralization

The process of modifying a schema so that it has identical data formats on each platform that runs the application. Neutralization overcomes the fact that different compilers lay out data in different ways.

See also *heterogeneous*.

object cluster

A fixed-size portion of a segment into which objects can be clustered when they are allocated.

See also *clustering*, *database segment*. Also see **os_object_cluster** in the *ObjectStore C++ API Reference*.

object faulting

The process by which dereferencing a pointer to a persistent object not yet available in virtual memory causes the object to be paged in.

See also *persistence*.

ObjectStore client

A client application that maps persistent database objects to virtual addresses, allocates and deallocates storage for persistent objects, maintains the cache of recently used pages and the lock status of those pages, and handles page faults on addresses that refer to persistent objects.

ObjectStore directory

Contains one or more rawfs databases, created by applications. It is *not* an operating system directory, but a directory in an ObjectStore file system.

See also *database*.

ObjectStore file system

Each disk used for storing rawfs databases contains an ObjectStore file system. Each file system is either a raw partition or an operating system file.

See also *database*, *ObjectStore Server*.

ObjectStore library

A library with which ObjectStore applications must link. Applications that use collections or inverses must also link with the collections library.

See also *collection*, *inverse*.

ObjectStore reference

An instance of one of the system-supplied reference classes. An ObjectStore reference can be used as a substitute for a pointer. References are always valid across databases and transactions.

See also *referent type*. Also see **os_Reference** or **os_reference** in the *ObjectStore C++ API Reference*.

ObjectStore Server

A daemon process that performs all access to an ObjectStore file system, including the storage and retrieval of persistent data. The system administrator typically starts a Server when ObjectStore is installed. If a site has more than one ObjectStore file system, additional Servers can be started to handle the additional file systems.

objectstore_ exceptionclass

Derived from the class **tix_exception**. Every built-in exception that ObjectStore can signal at run time is an instance of **objectstore_ exception**, and represents a particular type of error condition or exceptional circumstance.

See also *exception facility*, *tix_exception class*. Also see Appendix A, Exception Facility, in the *ObjectStore C++ API Reference*.

path

See *index path*.

path string

The character string supplied when you create an index path that specifies a sequence of member accesses.

See also *index path*, *type string*.

persistence

The ability to store data in such a way that its existence extends beyond the lifetime of the process that created it. ObjectStore supports persistence by providing C++ programs with direct, transparent access to ObjectStore databases.

See also *clustering*, *database entry point*, *navigational access*, *object faulting*, *persistent new*, *query processing*.

persistent data

Data that survives beyond the lifetime of the process that created it, that is, data stored in a database, as opposed to transient data.

See also *persistence*.

persistent delete

Persistent storage can be reclaimed using the **delete** operator, just as **persistent delete** operates when used for transient objects.

See also *persistence*, *persistent new*.

persistent memory

Database memory, as opposed to transient or program memory.

persistent new

There are several system-supplied overloadings of the C++ global function **operator new()** for creation of persistently allocated objects. These functions take one of two arguments that specify clustering information.

See also *persistence*.

persistent storage region

A client's persistent address space. This is a range of addresses that ObjectStore uses to store only persistent data.

See also *address space*.

process-local data member

Changes to a process-local data member remain in effect only for the duration of the process that made them, and are not visible to other processes.

query optimizer

Formulates efficient retrieval strategies, minimizing the number of objects examined in response to a query. The user enables optimization of queries over a given collection by adding indexes to the collection.

See also *collection*, *index*, *index path*, *query processing*.

query processing

Associative lookups of an object in the database, for example, by name or ID number.

See also *collection*, *index*, *navigational access*, *index path*, *query optimizer*.

query string

Used as an argument to a query function to express the selection criterion of a query.

See also *query processing*.

rank function

A user-defined function that determines an ordering for the instances of the class. The user must define a rank function whenever optimization is requested for range queries involving lookup of instances of a class.

See also *collection*, *endian types*, *iteration*, *query processing*, *query optimizer*.

rawfs

An ObjectStore raw file system containing ObjectStore directories and databases. A rawfs is made up of one or more files or raw partitions. It is independent of the file system managed by the operating system. Each ObjectStore Server can manage one rawfs.

See also *ObjectStore directory*.

rawfs database

A database that you store in an ObjectStore *rawfs*.

reachable types

The set of types related, through inheritance or embedding, to types that are marked persistent.

remote schema database

A database whose schema is stored in another database.

See also *database schema*.

recursive query

Performed by augmenting a collection during an iteration over that collection.

See also *collection*, *iteration*.

reference

See *ObjectStore reference*.

referent type

The referent type of an ObjectStore reference is the type of object the reference refers to.

See also *ObjectStore reference*.

root

See *database root*.

schema

A set of class definitions. A database's schema includes the definition of each class of object stored in the database. An application's schema includes the definition of each class the application retrieves from or allocates in persistent memory.

See also *application schema database*, *schema installation*, *schema validation*.

schema database

An ObjectStore database that contains a schema.

schema evolution The changes that a database's schema undergoes during the course of the database's existence, especially schema changes that potentially require changing the representation of objects already stored in the database.

See also *schema, instance migration*. Also see **os_schema_evolution** in the *ObjectStore C++ API Reference*.

schema generator ObjectStore Schema Generator (**ossg**) utility used to create application, compilation, and library schemas.

See also Overview of Schema Generation in *ObjectStore Building Applications*.

schema installation The process of supplementing a database's schema with classes from an application's schema. Installation can be performed either all at once (batch installation) or incrementally, at the user's discretion.

schema segment The segment of each ObjectStore database that contains the database's schema information. Users cannot allocate application objects in this segment.

See also *schema*, *schema installation*, *schema validation*.

schema source file The schema source file specifies the C++ classes that your code reads from or writes to persistent memory. You create the schema source file according to a specified format. See Chapter 3, Generating Schemas, in *ObjectStore Building Applications* for specific details.

schema validation The process of comparing a database schema and application schema to ensure that they are compatible.

See also *schema*.

segment See *database segment*.

serialization The sequence of transactions occurring in a particular operation.

Server host The host machine of an ObjectStore Server process.

set An unordered collection that has, at most, one occurrence of each element at a given time. Each set is an instance of the class **os_Set** or **os_set**, a class in the ObjectStore Class Library and part of the ObjectStore collection facility.

See also *bag*, *class extent*, *collection*, *collection cardinality*, *cursor*, *element*, *endian types*, *iteration*, *list*, *multivalued*, *query processing*.

subobject

An object contained in a given object, either as a data member value or as an object corresponding to a base class of the given object's direct class.

template instantiation

The process of type substitution of a parameterized type.

template specialization

The implementation of special (customized) functions for a particular template instantiation.

tix_exception class

Every user-defined or built-in exception is an instance of **objectstore_exception**, and represents a particular type of error condition or exceptional circumstance.

See also *exception facility*, *objectstore_exceptionclass*.

transaction

Transactions group-process database manipulation into atomic units.

See also *abort*, *commit*, *commseg*. Also see **os_transaction** in the *ObjectStore C++ API Reference*.

transient data

Data whose lifetime does not extend beyond the duration of the process that created it.

See also *persistence*, *persistent data*.

transient database

A system-supplied, pseudodatabase that can be supplied as an argument to **new** to create transient data.

See also *database*, *persistence*, *persistent new*, *transient segment*. Also see **os_database::get_transient_database()** in the *ObjectStore C++ API Reference*.

transient virtual memory

Program memory, as opposed to database memory.

See also *persistence*.

transient pointer

A transient object whose value is a pointer. A transient pointer to persistent storage is not valid across transaction boundaries, unless the member function **retain_persistent_addresses()** is used.

See also *transaction*, *transient data*.

transient segment

A system-supplied segment that can be supplied as an argument to persistent **new** to create transient data.

See also *persistence*, *persistent new*, *segment*, *transient data*, *transient database*.

type string

A character string specified when an index pathis created. The type string indicates the element type of collections for which the path can specify an index key or iteration order.

See also *index path*, *path string*.

union

A special instance of a class intended to save space.

value type

The value type of a data member is the declared type of the member's values.

virtual base class

A mechanism that allows a base class to appear multiple times in a derivation hierarchy.

VFT or vtbl

When a class is declared to have virtual functions or virtual base classes, the class acquires an invisible data member that points to a table of function pointers that are used by the application to call virtual functions. On some platforms these are known as VFTs and on others vtbls.

virtual memory

Virtual memory contains the application's data. Each application has its own virtual memory.

wired page

If a page is *wired* into the client cache, it is not considered as a candidate for cache replacement.

See also *client cache*, *cache replacement*, *eviction*.