NAME

manualindex – manual indexes in Shore

SYNOPSIS

```
enum IndexKind { LHash, BTree, UniqueBTree, RTree, RDTree };
struct IndexId {
      lrid_t           obj;
      int              i;
};
typedef IndexId IndexId;

VASResult       shore_vas::addIndex(
    const IndexId       &iid,
    IndexKind           indexKind
);

VASResult       shore_vas::dropIndex(
    const IndexId       &iid,
);

VASResult       shore_vas::statIndex(
    const IndexId       &iid,
);

VASResult       shore_vas::insertIndexElem(
    const IndexId       &iid,
    const vec_t         &key,
    const vec_t         &value
);
VASResult       shore_vas::removeIndexElem(
    const IndexId       &iid,
    const vec_t         &key,
    const vec_t         &value
);

VASResult       shore_vas::removeIndexElem(
    const IndexId       &iid,
    const vec_t         &key,
    int                 *numremoved // # elems removed
);

VASResult       shore_vas::findIndexElem(
    const IndexId       &iid,
    const vec_t         &key,
    const vec_t         &value,
    ObjectSize          *vallen,
    bool          *found
);
```

DESCRIPTION

Manual indexes in Shore are appendages to Shore objects. An object can have a fixed number of manual indexes. When the object is created, the number of indexes is given, and the SVAS allocates space in the object's metadata to store the given number of serial numbers for those indexes. The indexes are not created until the language binding layer calls the method **addIndex,** at which time an index is created and its serial number is placed in the object's metadata. The language binding may also call **dropIndex** to remove an index from an object. If an object with indexes is destroyed, all the manual indexes associated with the object are destroyed by the SVAS.

The language binding may call **insertIndexElem** and **removeIndexElem** to update indexes. When keys are known, the language binding layer locates values associated with the keys by calling **findIndexElem.** Indexes are scanned with a set of methods that are described in **indexscan(svas).**

The Shore Value-Added Server maintains an array of the logical identifiers of the indexes that are "contained" in an object. The array has origin 0, and by repeatedly calling **indexId,** the identifiers of the indexes in an object can be discovered.

The disk space used for an index and its entries is not reflected in the system properties of the containing object, but it can be computed by calling **statIndex** on each index contained in an object.

ARGUMENTS

The argument *iid* identifies the index by identifying the object that "owns" the index, and the ordinal number of the index for that object.

When an index is created, its type must be given in *indexKind.* There is no default index type. TODO add references for index types (btree, rtree, etc).

An index is part of an object, and when it is manipulated, the system properties of the "owning" object are checked for access permission. The object is thereby locked.

To insert a key-value pair into an index, both the the key and value are treated as byte-strings. They need not be located in contiguous memory the caller's address space, since vectors are used to describe both the key and value. InsertIndexElem does not change the vectors or the values identified by the vectors.

To remove one key-value pair, use the first form of **removeIndexElem,** in which the caller provides both the key and value. To remove all the values associated with a key in a non-unique index, use the second form of **removeIndexElem,** in which only the key is given. The SVAS returns the number of key-value pairs removed in ∗*numremoved. Numremoved* may not be a null pointer.

To find a value associated with a known key, use the method *findIndexElem.* The caller passes in the key and the value is returned. The value is placed in caller's address space in the place or places identified by the argument *value.* The caller is responsible for allocating this space, and it must be large enough to accommodate any of the values associated with the given key. The SVAS writes the length of the value returned into *vallen.* The vector *value* is passed as a *const reference* because the vector is not modified, even though *the memory areas that are identified by this vector are updated.* If *eof* is not null, **findIndexElement** writes **TRUE**(1) into ∗*eof* if there is a value associated with the given key. If there is no value for the key, it writes **FALSE**(0) in ∗*eof.*

ENVIRONMENT

All these methods are used in transactions, and can be invoked by clients and by value-added servers.

ERRORS

Deadlocks can occur while locks are being acquired. See **transaction(svas)** for information about deadlocks.

A complete list of errors is in **errors(svas).**

VERSION

This manual page applies to Version 1.1 of the Shore software.

SPONSORSHIP

The Shore project is sponsored by the Advanced Research Project Agency, ARPA order number 018 (formerly 8230), monitored by the U.S. Army Research Laboratory under contract DAAB07-91-C-Q518.

COPYRIGHT

SEE ALSO

registered(svas), transaction(svas), errors(svas) and **indexscan(svas).**