NAME

anonymous – creating and destroying anonymous Shore objects

SYNOPSIS

```
// create anonymous objects with initialized contents

VASResult
shore_vas::mkAnonymous(
    const lrid_t        &pool,   // oid of pool object
    const lrid_t        &typeobj,// object's type
    const vec_t         &core,   // data for core
    const vec_t         &heap,   // data for heap
    ObjectOffset        tstart,  // where text starts
    int                 nindexes, // # indexes
    lrid_t              *resultref,
    void                *physid=0
);

VASResult
shore_vas::mkAnonymous(
    const lrid_t        &pool,   // oid of pool object
    const lrid_t        &typeobj,// object's type
    const vec_t         &core,   // data for core
    const vec_t         &heap,   // data for heap
    ObjectOffset        tstart,  // where text starts
    int                 nindexes, // # indexes
    const lrid_t        ref      // preallocated serial#
);

// create anonymous objects with uninitialized contents

VASResult
shore_vas::mkAnonymous(
    const lrid_t        &pool,   // oid of pool object
    const lrid_t        &typeobj,// object's type
    ObjectSize          csize,   // size of core
    ObjectSize          hsize,   // size of heap
    ObjectOffset        tstart,  // where text starts
    int                 nindexes, // # indexes
    lrid_t              *resultref
);

VASResult
shore_vas::mkAnonymous(
    const lrid_t        &pool,   // oid of pool object
    const lrid_t        &typeobj,// object's type
    ObjectSize          csize,   // sizeo of core
    ObjectSize          hsize,   // sizeo of heap
    ObjectOffset        tstart,  // where text starts
    int                 nindexes, // # indexes
    const lrid_t        ref      // preallocated serial#
);

VASResult       shore_vas::rmAnonymous(
```

```
        const lrid_t        &obj
    );
```

DESCRIPTION

These methods create and destroy anonymous objects, which are objects that reside in a **pool.** The caller is responsible for maintaining the integrity of the objects' types, which means that the caller must update the inverses of bidirectional relationships and indexes in which the object in question participates.

These methods *cannot* be used to create indexes.

ARGUMENTS

When an anonymous object is created, its location (pool) must be identified by an object identifier.

The argument *typeobj* must refer to a type in a frozen **module.**

The first and second forms of the method **mkAnonymous** create objects with initialized data; the third and fourth forms create objects with uninitialized data. When initialized data are given (arguments *core* and *vec*), the vectors may be of any size; they may also be empty. Once an object is created, its core size cannot change.

The first and third forms of the method **mkAnonymous** create objects without requiring a pre-allocated serial number. When these methods are used, the server allocates and returns a serial number. The argument *resultref* is used both for passing information in and out of the methods. On input, the value of ∗*resultref* is inspected; if its serial number is null (in the sense that

```
(∗resultref) == serial_t::null
```

-- see **oid(shore)**), the Shore Value-Added Server allocates a new identifier for the object to be created. If the serial number is not null, the value ∗*resultref* is considered to be a pre-allocated, unused object identifier, and it is given to the object to be created. In both cases, the logical object identifier for the object created is returned in ∗*resultref.*

The second and fourth forms of the method **mkAnonymous** create objects with a pre-allocated serial number, *ref.*

Tstart specifies the first byte (from the beginning of the object) of the TEXT attribute. If the object has no TEXT attribute, the value of *tstart* must be **NoText.** (This is not checked by the Value-Added Server. It is the responsibility of the caller to see that this is the case.)

The argument *nindexes* is the number of manual indexes that the object will have when the type system has finished creating the object. The SVAS keeps track of the indexes so that it can destroy them when the object is destroyed. This value cannot be changed after the object is created.

In the first form of **mkAnonymous,** the argument *physid* is optional, and if present, it points to a place in the caller's addresss space into which the SVAS will place the physical object identifier of the object that was created. This is for use by other Value-Added Servers that store physical object identifiers for performance reasons, and it is implemented only in the server libary; client processes may not use this feature.

ERRORS

If the caller provides a pre-allocated object identifier to **mkAnonymous,** the burden is on the caller to see that the object identifier is legitimate and is not already in use. The best way to do that is to allocate the object identifier and create the object in the same transaction.

The SVAS does require that the object identifier given be on the same volume as the pool in which the object is to be created. If it is not, method returns in failure, with the error status **ST_VolumesDontMatch.**

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

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

ENVIRONMENT

All these methods are available on the client side and the server side. The argument *physid* may not be used by a client.

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

pool(svas), manualindex(svas), indexscan(svas), oid(shore), errors(svas) and **text(svas).**