Standard ECMA-227

October 1995

ECMA

Standardizing Information and Communication Systems

Portable Common Tool Environment (PCTE) - Extensions for support of Fine-Grain Objects - Abstract Specification

Phone: +41 22 849.60.00 - **Fax:** +41 22 849.60.01 - **URL:** http://www.ecma.ch - **Internet:** helpdesk@ecma.ch

Standard ECMA-227
October 1995

ECMA

Standardizing Information and Communication Systems

Portable Common Tool Environment (PCTE) - Extensions for support of Fine-Grain Objects - Abstract Specification

Brief History

Software engineering tools are increasingly manipulating large amounts of objects accessed by several application developers in the context of integrated software development environments. With PCTE, defined in Standard ECMA-149, the software community has all the basic functionalities required to develop such repositories. In early 1993, however, it became apparent that not all objects manipulated by software tools need the same level of flexibility but, on the other hand, very often require performance which is hard to achieve with all the properties associated with PCTE objects in general. Typically, a tool needs to manipulate a large set of objects which are mostly accessed at the same time (therefore allowing simplified concurrent access mechanisms), with very short access times.

In 1993, several projects addressed this problem. Two of them produced results which were made publicly available and were thereafter used as input to this Standard:

- the Portable Common Interface Set (PCIS) project of the NATO Special Working Group on APSE,
- the Object Oriented Tool Interface Set (OOTIS) project of IBM.

By the end of 1993, the US Department of Defense, the US National Institute of Standards and Technology (NIST), and the Object Management Group (OMG) decided to create an initiative, called the North American PCTE Initiative (NAPI) in order to resolve this problem (among others).

At the same time, the technical committee TC33 of ECMA decided to create a new task group, named TGOO, to add object orientation and support of fine-grain objects to PCTE. The NAPI and TGOO groups soon decided to merge their efforts in order to prepare a joint specification.

In 1994, the NAPI group transformed itself into the OMG Special Interest Group on PCTE (OMG PCTE SIG) and the joint work with ECMA TC33-TGOO continued.

In September 1994, a new working group ISO/IEC JTC1/SC22/WG22 was created to manage the maintenance of the PCTE International Standard ISO/IEC 13719, which is equivalent to Standard ECMA-149 3rd edition. That working group participated in the review of the final drafts of this Standard.

This Standard is the result of all these collaborative efforts.

This ECMA Standard has been adopted by the ECMA General Assembly in October 1995.

Table of contents

1 Scope

(1) This ECMA Standard specifies fine-grain extensions to PCTE, as defined in Standard ECMA-149.

(2) The extensions described in this Standard are as follows:

(3) • For some operations specified in ECMA-149, the semantics is extended for the cases when the objects manipulated by these operations are fine-grain objects. Some operations have their semantics affected because new errors may occur.

(4) • There are new operations.

(5) • There are new errors.

(6) When objects passed as arguments of PCTE operations are not fine-grain objects, the syntax and semantics of the operations described in this Standard are the same as the syntax and semantics of operations described in ECMA-149. For this reason, this Standard is said to be *upwards compatible* from ECMA-149.

2 Conformance

2.1 Conformance of binding

(1) The provisions of 2.1 of ECMA-149 are extended to cover the operations, datatypes and error values of this Standard.

2.2 Conformance of implementation

(1) The functionality of PCTE is divided into several modules, as defined in ECMA-149, clause 2.2. This Standard introduces a new module: the *fine-grain objects module*, which consists of all the extensions specified in this Standard and defines a new conformance level of PCTE implementations.

(2) An implementation of PCTE conforms to ECMA-149 with fine-grain objects if and only if it implements the core module and, in addition, implements the fine-grain objects module.

3 Normative references

(1) The following Standard contains provisions which, through reference in this text, constitute provisions of this Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this Standard are encouraged to investigate the possibility of applying the most recent editions of the Standard indicated below.

(2)	ECMA-149	Portable Common Tool Environment (PCTE) - Abstract Specification (3rd edition, December 1994)

4 Definitions

4.1 Technical terms

(1) All technical terms used in this Standard, other than a few in widespread use, are defined in the text, usually in a formal notation, or in ECMA-149. All identifiers defined in VDM-SL or in DDL (see ECMA-149, 4.1) are technical terms; apart from those, a defined technical term is printed in italics at the point of its definition, and only there.

4.2 Other terms

(1) For the purpose of this Standard, all non-technical terms defined in 4.2 of ECMA-149 apply.

5 Formal notations

(1) The formal notations used in this Standard are those defined in clause 5 of ECMA-149.

6 Overview of support of fine-grain objects in PCTE

(1) The notion of support of fine-grain objects is mostly concerned with improved performance time for creating and accessing PCTE objects. Object granularity is not dependent on type. It is described in terms of the amount of processing that has to be done to access an object.

(2) To enhance performance, the concept of cluster is introduced. A *cluster* is an object that represents the set of fine-grain objects that share the same values for certain PCTE properties and with some specific restrictions:

(3) • Usage restrictions on concurrency allow them to be cached in the main memory of processes.

(4) • Time attributes of all fine-grain objects residing in a cluster are shared.

(5) • Notification is not applicable to fine-grain objects.

(6) • Security properties are also shared and only checked once at the level of the cluster.

(7) • Auditing has limitations which decreases the controls to be made on fine-grain objects.

(8) • Fine-grain objects are not accountable resources.

(9) • Fine-grain objects have the same replicated state as their cluster.

7 Outline of the Standard

(1) Clause 6 gives an informal, non-normative explanation of the concepts of support of fine-grain objects. Clause 7 gives an overview of the document and of the structure of the definition.

(2) This Standard follows the same structure as ECMA-149: all clauses are normative except clauses 6 and 7 and annex E which are informative. For each clause of this Standard, an introduction summarizes the extensions made to the corresponding clause of ECMA-149 in order to support fine-grain objects. Then the semantic changes of affected operations (if any) are described, and finally new operations (if any) are specified.

8 Foundation

(1) The foundations described in clause 8 of ECMA-149 are not affected by the support of fine-grain objects.

9 Object management

9.1 Object management concepts

(1) **sds** system:

(2) **extend** object **with**
attribute
    cluster_identifier: **(read) non_duplicated natural**;
end object;

(3) **end** system;

(4) The cluster identifier identifies the cluster in which the object resides. If the cluster identifier is 0, the object does not reside in a cluster. If the cluster identifier is not 0, it is the key of a "known_cluster" link from the volume on which the object resides to the cluster in which the object resides. See 11.1.

(5) An object which resides in a cluster is called a *fine-grain* object. An object which does not reside in a cluster is called a *coarse-grain* object. The same object can be created as coarse-grain object and become fine-grain after it is moved into a cluster and conversely.

(6) Objects which have the following types (or one of their descendant types) cannot reside in a cluster. They are always coarse-grain objects:

(7) • "file", "pipe", "message_queue", "device", "accounting_log", "audit_file";

(8) • "volume", "cluster", "archive", "archive_directory";

(9) • "process", "activity";

(10) • "common_root";

(11) • "sds";

(12) • "workstation", "execution_class", "execution_site", "execution_site_directory";

(13) • "replica_set_directory", "replica_set";

(14) • "security_group", "program_group", "mandatory_directory", "mandatory_class", "security_group_directory";

(15) • "accounting_directory", "consumer_group", "resource_group".

(16) The last access time of a fine-grain object is equal to the default initial value of time attributes (see 8.3.2 of ECMA-149).

(17) The last modification time of a fine-grain object is equal to the last modification time of the cluster in which it resides.

(18) The last change time of a fine-grain object is equal to the last change time of the cluster in which it resides.

(19) The last modification time of a fine-grain object is set only:

(20) • when the object is created in a cluster (operations OBJECT_CREATE, OBJECT_COPY, VERSION_REVISE, VERSION_SNAPSHOT),

(21) • when the object is moved into a cluster (operation OBJECT_MOVE),

(22) • when the last modification time of the cluster in which it resides is modified.

(23) The last modification time of a cluster is the system time of the last release of a write or delete lock for an object residing in the cluster.

(24) The replicated state of a fine-grain object is equal to the replicated state of the cluster in which it resides.

(25) *NOTE - Neither a fine-grain object nor a cluster has contents, so the last access time is meaningless for both. That is why it is always equal to the default initial value of time attributes.*

9.2 Link operations affected by support of fine-grain objects

9.2.1 LINK_CREATE

(1)
```
LINK_CREATE (
    origin          : Object_designator,
    new_link        : Link_designator,
    dest            : Object_designator,
    reverse_key     : [ Actual_key ]
)
```

New semantics

(2) If *dest* is a fine-grain object and *new_link* is a composition link, then any security group that has OWNER granted or denied to *origin* has OWNER granted or denied respectively to all objects which reside in the cluster of *dest*; similarly if *origin* is a fine-grain object and *reverse_link* (the reverse link of *new_link*) is a composition link, then any security group that has OWNER granted or denied to *dest* has OWNER granted or denied respectively to all objects which reside in the cluster of *origin*. This requires the process to have OWNER rights on *dest* or *origin* respectively. See 19.1.2 of ECMA-149 for details.

9.2.2 LINK_DELETE

(1)
```
LINK_DELETE (
    origin      : Object_designator,
    link        : Link_designator
)
```

New semantics

(2) For each deleted fine-grain object *object*, the "object_in_cluster" link from the cluster in which *object* was residing to *object* is also deleted.

9.2.3 LINK_REPLACE

(1) LINK_REPLACE (

```
    origin              : Object_designator,
    link                : Link_designator,
    new_origin          : Object_designator,
    new_link            : Link_designator,
    on_reverse_key      : [ Actual_key ]
)
```

New semantics

(2) The semantics of this operation refers to LINK_DELETE. It is therefore affected in the same way.

9.3 Object operations affected by support of fine-grain objects

9.3.1 OBJECT_COPY

(1) OBJECT_COPY (

```
    object              : Object_designator,
    new_origin          : Object_designator,
    new_link            : Link_designator,
    reverse_key         : [ Actual_key ],
    on_same_volume_as   : [ Object_designator ],
    access_mask         : Atomic_access_rights
)
    new_object          : Object_designator
```

New semantics

(2) If *on_same_volume_as* is supplied, then *new_object* and all its components reside in the same volume as *on_same_volume_as*, as currently specified in ECMA-149.

(3) If *on_same_volume_as* is not supplied, then *new_object* resides in the same volume as *object*, and each component of *new_object* resides in the same volume as its corresponding component in *object*, as currently specified in ECMA-149.

(4) Additionally:

(5)
- If *on_same_volume_as* is supplied and if the cluster identifier of *on_same_volume_as* is not 0, then *new_object* and all its components reside in the same cluster as *on_same_volume_as*.

(6)
- If *on_same_volume_as* is supplied is itself a cluster, then *new_object* and all its components reside in the cluster *on_same_volume_as*.

(7)
- If *on_same_volume_as* is not supplied and if the cluster identifier of *object* is not 0, then *new_object* resides in the same cluster as *object*. Similarly, if the cluster identifier of a component of *object* is not 0, the corresponding component of *new_object* is created in the same cluster as that component of *object*, and so on for subcomponents.

(8) If *new_object* or any of its components is created in a cluster, then an "object_in_cluster" link is created from that cluster to the new object or component. Each created link is keyed by the exact identifier of the created object.

(9) For each object X created in a cluster C:

(10)
- The atomic ACL of X is set to the atomic ACL of C.

(11)
- The security labels of X are set to the security labels of C.

(12)
- The last modification time and last change time of X are set to the last modification time and last change time of C, respectively.

New errors

(13) OBJECT_CANNOT_BE_CLUSTERED (*object* or its components)

(14) If *object* is a fine-grain object:
ACCESS_ERRORS (cluster of *object*, ATOMIC, MODIFY, APPEND_LINKS)

9.3.2 OBJECT_CREATE

(1) OBJECT_CREATE (

type	: Object_type_nominator,
new_origin	: Object_designator,
new_link	: Link_designator,
reverse_key	: [Actual_key],
on_same_volume_as	: [Object_designator],
access_mask	: Atomic_access_rights

)

new_object	: Object_designator

New semantics

(2) If *on_same_volume_as* is supplied, then *new_object* resides in the same volume as *on_same_volume_as*, as currently specified in ECMA-149.

(3) If *on_same_volume_as* is not supplied, then *new_object* resides in the same volume as *new_origin*, as currently specified in ECMA-149.

(4) Additionally:

(5) • If *on_same_volume_as* is supplied and if the cluster identifier of *on_same_volume_as* is not 0, then *new_object* resides in the same cluster as *on_same_volume_as*.

(6) • If *on_same_volume_as* is itself a cluster, then *new_object* resides in the cluster *on_same_volume_as*.

(7) • If *on_same_volume_as* is not supplied and if the "cluster_identifier" of *new_origin* is not 0, then *new_object* resides in the same cluster as *new_origin*.

(8) If *new_object* is created in a cluster, then:

(9) • An "object_in_cluster" link is created from this cluster to the new object. The created link is keyed by the exact identifier of the created object.

(10) • The atomic ACL of *new_object* is set to the atomic ACL of the cluster.

(11) • The security labels of *new_object* are set to the security labels of the cluster.

(12) • The last modification time and last change time of *new_object* are set to the last modification time and last change time of the cluster, respectively.

New errors

(13) OBJECT_CANNOT_BE_CLUSTERED (object to be created)

(14) If *object* is a fine-grain object:
ACCESS_ERRORS (cluster of *object*, ATOMIC, MODIFY, APPEND_LINKS)

9.3.3 OBJECT_DELETE

(1) OBJECT_DELETE (

origin	: Object_designator,
link	: Link_designator

)

New semantics

(2) For each deleted fine-grain object, if any, the "object_in_cluster" link from the cluster in which the deleted object resided to the deleted object is also deleted.

New errors

(3) If any deleted object is a fine-grain object:
ACCESS_ERRORS (cluster of deleted object, ATOMIC, MODIFY, WRITE_LINKS)

9.3.4 OBJECT_MOVE

(1)
```
OBJECT_MOVE (
    object                  : Object_designator,
    on_same_volume_as       : Object_designator,
    scope                   : Object_scope
)
```

New semantics

(2) *object* and all its components are moved to the same volume as *on_same_volume_as*, as currently specified in ECMA-149.

(3) Additionally, if the cluster identifier of *on_same_volume_as* is not 0, or *on_same_volume_as* is itself a cluster, then:

(4) - *object* and all its components are moved into the cluster of *on_same_volume_as*, or the cluster *on_same_volume_as*.

(5) - An "object_in_cluster" link is created from that cluster to the moved object and to each of its components and subcomponents. Each created link is keyed by the exact identifier of the moved object.

(6) - The atomic ACLs of *object* and all its components are set to the atomic ACL of the cluster.

(7) - The security labels of *object* are set to the security labels of the cluster.

(8) - The last modification time and last change time of *object* and all its components are set to the last modification time and last change time of the cluster, respectively.

(9) For *object* (if moved) and each moved component, which was previously residing in a cluster, the "object_in_cluster" link from the cluster to the object is deleted.

New errors

(10) OBJECT_CANNOT_BE_CLUSTERED (*object*)

(11) If *object* is a fine-grain object:
ACCESS_ERRORS (cluster of *object*, ATOMIC, MODIFY, WRITE_LINKS)

(12) If *on_same_volume_as* resides in or is a cluster *cluster*:
ACCESS_ERRORS (*cluster*, ATOMIC, MODIFY, APPEND_LINKS)

9.3.5 OBJECT_SET_TIME_ATTRIBUTES

(1)
```
OBJECT_SET_TIME_ATTRIBUTES (
    object                  : Object_designator,
    last_access             : [ Time ],
    last_modification       : [ Time ],
    scope                   : Object_scope
)
```

New semantics

(2) If *object* is a cluster, the new time attributes are also set on all objects residing in the cluster.

New errors

(3) If *object* is a fine-grain object:
OBJECT_IS_FINE_GRAIN (*object*)

9.4 Version operations affected by support of fine-grain objects

9.4.1 VERSION_IS_CHANGED

(1)
```
VERSION_IS_CHANGED (
    version         : Object_designator,
    predecessor     : Natural
)
    changed         : Boolean
```

New errors

(2) If *object* is a fine-grain object:
OBJECT_IS_FINE_GRAIN (*object*)

9.4.2 VERSION_REVISE

(1)
```
VERSION_REVISE (
    version                         : Object_designator,
    new_origin                      : Object_designator,
    new_link                        : Link_designator,
    on_same_volume_as               : [ Object_designator ],
    access_mask                     : Atomic_access_rights
)
    new_version                     : Object_designator
```

New semantics

(2) The semantics of VERSION_REVISE refers to the semantics of OBJECT_COPY. It is therefore indirectly changed in the same way.

9.4.3 VERSION_SNAPSHOT

(1)
```
VERSION_SNAPSHOT (
    version                         : Object_designator,
    new_link_and_origin             : [ Link_descriptor ],
    on_same_volume_as               : [ Object_designator ],
    access_mask                     : Atomic_access_rights
)
    new_version                     : Object_designator
```

New semantics

(2) The semantics of VERSION_SNAPSHOT refers to the semantics of OBJECT_COPY. It is therefore indirectly changed in the same way.

10 Schema management

(1) This clause is not affected by support of fine-grain objects.

11 Volumes, clusters, devices, and archives

11.1 Cluster concepts

(1) Cluster_identifier = Natural

(2) **sds** system:

(3)
```
extend object type volume with
link
    known_cluster: (navigate) non_duplicated existence link (cluster_identifier) to
            cluster reverse cluster_in_volume;
end volume;
```

(4)
```
cluster: child type of object with
attribute
    cluster_characteristics: (read) string;
link
    object_in_cluster: (navigate) non_duplicated designation link (exact_identifier) to
            object;
    cluster_in_volume: (navigate) implicit link to volume reverse known_cluster;
end cluster;
```

(5) **end** system;

(6) A cluster is an object which groups a set of objects sharing some common properties or behaviour in respect with concurrency control, time attributes, auditing, security, notification and accounting. See 9.1.

(7) The destinations of the "known_cluster" links from a volume are the clusters residing on that volume.

(8) The destinations of the "object_in_cluster" links from a cluster are called the objects *residing in* that cluster. The value of the "exact_identifier" attribute is the exact identifier of the object (see 9.1.1 of ECMA-149).

(9) All objects which reside in a cluster must also reside on the same volume as the volume of the cluster itself.

(10) The "cluster_characteristics" attribute is an implementation-defined string specifying implementation-dependent characteristics of the cluster.

11.2 Archive operations affected by support of fine-grain objects

11.2.1 ARCHIVE_RESTORE

(1)
```
ARCHIVE_RESTORE (
    device                  : Device_designator,
    archive                 : Archive_designator,
    scope                   : Archive_selection,
    on_same_volume_as       : Object_designator
)
    restoring_status        : Archive_status
```

New semantics

(2) Additionally, if the cluster identifier of *on_same_volume_as* is not 0, or if *on_same_volume_as* is itself a cluster, then for each restored object:

(3) • The object is allocated in the cluster of *on_same_volume_as*, or the cluster *on_same_volume_as*.

(4) • An "object_in_cluster" link is created from that cluster to the restored object. Each created link is keyed by the exact identifier of the restored object.

(5) • The atomic ACLs of *object* and all its components are set to the atomic ACL of the cluster.

(6) • The last access time, last modification time, and last change time of *object* and all its components are set to the last access time, last modification time, and last change time of the cluster, respectively.

New errors

(7) OBJECT_CANNOT_BE_CLUSTERED (any object being restored)

(8) If *on_same_volume_as* resides in or is a cluster *cluster*:
ACCESS_ERRORS (*cluster*, ATOMIC, MODIFY, APPEND_LINKS)

11.2.2 ARCHIVE_SAVE

(1)
```
ARCHIVE_SAVE (
    device              : Device_designator,
    archive             : Archive_designator,
    objects             : Object_designators
)
    archiving_status : Archive_status
```

New semantics

(2) For each object to be archived which resides in a cluster, the "object_in_cluster" link from the cluster on which the object resides to the object is deleted.

New errors

(3) For any object X of *objects* which resides in a cluster
ACCESS_ERRORS (cluster of X, ATOMIC, MODIFY, APPEND_LINKS)

11.3 New operations on clusters

11.3.1 CLUSTER_CREATE

(1)

```
CLUSTER_CREATE (
    on_same_volume_as          : Object_designator,
    cluster_identifier         : Natural,
    access_mask                : Atomic_access_rights,
    cluster_characteristics    : String

    )
    new_cluster                : Cluster_designator
```

(2) CLUSTER_CREATE creates a new cluster *new_cluster* in the volume *volume* in which the object *on_same_volume_as* resides.

(3) A new "known_cluster" link with key *cluster_identifier* is created from *volume* to *new_cluster*.

(4) *access_mask* is used in conjunction with the default atomic ACL and default object owner of the calling process to define the atomic ACL and the composite ACL which are to be associated with the created object (see 19.1.4 in ECMA-149).

(5) The confidentiality and integrity labels of *cluster* are respectively set to the confidentiality and integrity labels of the mandatory context of the calling process.

(6) The "cluster_characteristics" attribute of *new_cluster* is set to *cluster_characteristics*.

(7) Write locks of the default mode are obtained on *on_same_volume_as*, on *new_cluster*, and on the new "known_cluster" link.

Errors

(8) ACCESS_ERRORS (*volume*, ATOMIC, MODIFY, APPEND_LINKS)

(9) LIMIT_WOULD_BE_EXCEEDED (MAX_KEY_VALUE)

(10) OBJECT_OWNER_VALUE_WOULD_BE_INCONSISTENT_WITH_ATOMIC_ACL

(11) REFERENCE_CANNOT_BE_ALLOCATED

(12) CLUSTER_EXISTS (*cluster_identifier*, *volume*)

11.3.2 CLUSTER_DELETE

(1)

```
CLUSTER_DELETE (
    cluster   : Cluster_designator
)
```

(2) CLUSTER_DELETE deletes the "known_cluster" link to *cluster* from the volume *volume* on which *cluster* is residing. and then deletes *cluster*.

(3) Write locks (of the default kind) are obtained on *cluster* and the deleted cluster and the deleted link.

Errors

(4) ACCESS_ERRORS (*volume*, ATOMIC, MODIFY, WRITE_LINKS)

(5) ACCESS_ERRORS (*cluster*, ATOMIC, CHANGE, WRITE_IMPLICIT)

(6) If the conditions hold for deletion of the "cluster" object *cluster*:
    ACCESS_ERRORS (*volume*, ATOMIC, MODIFY, DELETE)

(7) CLUSTER_HAS_OTHER_LINKS (*cluster*)

(8) CLUSTER_IS_UNKNOWN (*cluster*)

11.3.3 CLUSTER_LIST_OBJECTS

(1)
```
CLUSTER_LIST_OBJECTS (
    cluster  : Cluster_designator,
    types    : Object_type_nominators
)
    objects  : Object_designators
```

(2) CLUSTER_LIST_OBJECTS returns in *objects* a set of object designators determined by *types*.

(3) An object designator is returned in *objects* for each object which resides in *cluster*, whose type in working schema is an element of *types*.

(4) A read lock of the default mode is obtained on *cluster*.

Errors

(5) ACCESS_ERRORS (*cluster*, ATOMIC, READ, READ_LINKS)

(6) REFERENCE_CANNOT_BE_ALLOCATED

12 Files, pipes, and devices

(1) This clause is not affected by support of fine-grain objects.

(2) *NOTE - Files, pipes, and devices cannot reside in clusters, see clause 9.*

13 Process execution

(1) This clause is not affected by support of fine-grain objects.

(2) *NOTE - It is intended that a process loads all objects of the clusters that it is accessing. The conditions allowing the cache to be loaded and downloaded are related to the locking policies. See clause 16.*

14 Message queues

(1) This clause is not affected by support of fine-grain objects.

(2) *NOTE - Message queues cannot reside in clusters, see clause 9.*

15 Notification

15.1 Notification concepts

(1) Notifiers cannot be associated with fine-grain objects.

15.2 Notification operations affected by support of fine-grain objects

15.2.1 NOTIFY_CREATE

(1)
```
NOTIFY_CREATE (
    notifier_key   : Natural,
    queue          : Message_queue_designator,
    object         : Object_designator
)
```

New errors

(2) OBJECT_IS_FINE_GRAIN (*object*)

16 Concurrency and integrity control

(1) When an activity acquires a lock on a fine-grain object, the locking request is done on the "cluster" in which the object resides.

(2) When an activity requests a lock on a cluster, the lock is acquired only if all the following conditions hold:

(3) • The requested external lock mode is compatible with the external lock mode of other locks obtained by concurrent activities, as in ECMA-149.

(4) • The requested internal lock mode is compatible with the external lock mode of the child activities, as in ECMA-149.

(5) • The requested external lock mode is compatible with the internal lock mode of the parent activity (if any), as in ECMA-149.

(6) • If a read lock is already acquired by at least one different process running in the same activity and the current process is performing an operation which requests a write lock, the lock acquisition (such a request is a promotion from read to write for a coarse-grain object) is delayed until the lock can be promoted to write and until all other processes which have made a lock request are terminated.

(7) • If a write lock is already acquired by one (and only one) different process running in the same activity and the current process is performing an operation which requests a read or a write lock acquisition on a fine-grain object (such a request is necessarily satisfied in case of a lock on a coarse-grain object), the lock acquisition is delayed until the process which made the write lock request on the cluster is terminated.

NOTES

(8) *1. It is intended that an implementation supports caching of fine-grain objects by loading in main memory all the objects of a cluster. For performance reasons, it is intended that the loading of all the objects of a cluster is done in the private user space of processes which need to access the objects.*

(9) *2. The additional locking rules prevent two different processes running in the same activity from accessing the same fine-grain objects and from performing concurrent non-synchronized updates on their caches. With these additional locking rules, the loading of the cache is intended to happen as follows:*

- *when an activity acquires a read lock on a cluster, all objects of the cluster are placed in a read-only cache stored in the space of the process which is performing the operation causing the lock acquisition.*
- *When an activity acquires a write lock on a cluster, all objects of the cluster are placed in a read-write cache stored in the space of the process which is performing the operation causing the lock acquisition.*
- *Several processes can have read-only caches on the same cluster.*
- *Only one process can have a read-write cache on a cluster, at a given time.*

(10) *3. It is intended that a process unloads a cache when the activity causing the cache to be loaded is ended or aborted. If the activity commits, the cache has to be downloaded to the object base. If the activity is aborted, the cache must be simply discarded, without updates in the object base.*

(11) *4. Whenever this Standard or ECMA-149 says 'a read/write lock of the default mode is obtained on an object object', if object is a fine-grain object this is implicitly equivalent to 'a read/write lock of the default mode is obtained on the cluster in which object resides' as a consequence of the first rule above.*

17 Replication

17.1 Replication concepts

(1) When a cluster is duplicated, all the fine-grain objects residing in the cluster are replicated.

(2) A fine-grain object cannot be replicated in isolation (i.e. the only way to duplicate it is by duplicating its cluster).

17.2 Replication operations affected by support of fine-grain objects

17.2.1 REPLICATED_OBJECT_CREATE

(1) REPLICATED_OBJECT_CREATE (
replica_set : Replica_set_designator,
object : Object_designator
)

New semantics

(2) If *object* is a cluster, its replicated state is set to MASTER.

New errors

(3) OBJECT_IS_FINE_GRAIN (*object*)

17.2.2 REPLICATED_OBJECT_DUPLICATE

(1) REPLICATED_OBJECT_DUPLICATE (
object : Object_designator,
volume : Administration_volume_designator,
copy_volume : Administration_volume_designator
)

New semantics

(2) If *object* is a cluster, then all the objects which reside in the cluster are replicated

17.2.3 REPLICATED_OBJECT_REMOVE

(1) REPLICATED_OBJECT_REMOVE (
object : Object_designator
)

New semantics

(2) If *object* is a cluster, its replicated state is set to NORMAL.

New errors

(3) OBJECT_IS_FINE_GRAIN (*object*)

18 Network connection

(1) This clause is not affected by support of fine-grain objects.

19 Discretionary security

19.1 Concepts of discretionary security

(1) All fine-grain objects residing in a cluster have the same ACLs as the cluster.

19.2 Discretionary access control operations affected by support of fine-grain objects

19.2.1 OBJECT_SET_ACL_ENTRY

(1) OBJECT_SET_ACL_ENTRY (
object : Object_designator,
group : Group_identifier,
modes : Atomic_access_rights,
scope : Object_scope
)

New semantics

(2) If *object* is a cluster, then the same ACL entry is added to the ACL of all objects residing in the cluster.

New errors

(3) OBJECT_IS_FINE_GRAIN (*object*)

20 Mandatory security

20.1 Mandatory security concepts

(1) All fine-grain objects residing in a cluster have the same confidentiality and integrity labels as the cluster.

20.2 Mandatory security operations affected by support of fine-grain objects

20.2.1 OBJECT_SET_CONFIDENTIALITY_LABEL

(1)
```
OBJECT_SET_CONFIDENTIALITY_LABEL (
    object   : Object_designator,
    label    : Security_label
)
```

New semantics

(2) If *object* is a cluster, then the same confidentiality label is set on all objects residing in the cluster.

New errors

(3) OBJECT_IS_FINE_GRAIN (*object*)

20.2.2 OBJECT_SET_INTEGRITY_LABEL

(1)
```
OBJECT_SET_INTEGRITY_LABEL (
    object   : Object_designator,
    label    : Security_label
)
```

New semantics

(2) If *object* is a cluster, then the same confidentiality label is set on all objects residing in the cluster.

New errors

(3) OBJECT_IS_FINE_GRAIN (*object*)

21 Auditing

21.1 Auditing concepts

(1) Operations on fine-grain objects do not produce auditable events.

21.2 Auditing operations affected by support of fine-grain objects

21.2.1 AUDIT_ADD_CRITERION

(1)
```
AUDIT_ADD_CRITERION (
    station          : Workstation_designator,
    criterion        : Selection_criterion
)
```

New errors

(2) OBJECT_IS_FINE_GRAIN (*object*)

22 Accounting

(1) This clause is not affected by support of fine-grain objects.

(2) *NOTE - Accountable resources are files, pipes, volumes, devices, static contexts, message queues, and workstations none of which can be fine-grain objects.*

23 Common binding features

(1) This clause is not affected by support of fine-grain objects.

24 Implementation limits

(1) This clause is not affected by support of fine-grain objects.

Annex A (normative) - VDM Specification Language for the Abstract Specification

(1) This annex is not affected by support of fine-grain objects.

Annex B (normative) - The Data Definition Language (DDL)

(1) This annex is not affected by support of fine-grain objects.

Annex C (normative) - Specification of new errors

C.1 Access errors

(1) Additionally, ACCESS_ERRORS (*object*, *scope*, *access_mode*, *permission*) is extended for all access modes with the following:

(2) if object resides in a cluster then:
ACCESS_ERRORS (cluster of *object*, *scope*, *access_mode*, *permission*)

C.2 New errors

(1) OBJECT_IS_FINE_GRAIN (*object*)
object is fine-grain and an attempt is being made to perform one of the operations which are not permitted on fine-grain objects.

(2) CLUSTER_EXISTS (*cluster_identifier*, *volume*)
The specified cluster number *cluster_identifier* corresponds to an existing cluster in the volume *volume*.

(3) CLUSTER_HAS_OTHER_LINKS (*cluster*)
There are links starting from the cluster *volume* which are not the "cluster_in_volume" link to its associated volume.

(4) CLUSTER_IS_UNKNOWN (*volume*)
The "cluster" object *cluster* is not linked to a volume via link of type "known_cluster".

Annex D (normative) - Auditable events

(1) This clause is not affected by support of fine-grain objects.

Annex E (informative) - The predefined schema definition sets

(1) The fine-grain objects module requires the following extensions to the system SDS:

(2)
```
sds system:
```

(3)
```
extend object with
attribute
    cluster_identifier: (read) non_duplicated natural;
end object;
```

(4)
```
extend object type volume with
link
    known_cluster: (navigate) non_duplicated existence link (cluster_identifier) to
        cluster reverse cluster_in_volume;
end volume;
```

(5)
```
cluster: child type of object with
attribute
    cluster_characteristics: (read) string;
link
    object_in_cluster: (navigate) non_duplicated designation link (exact_identifier) to
        object;
    cluster_in_volume: (navigate) implicit link to volume reverse known_cluster;
end cluster;
```

(6)
```
    end system;
```