Mica Working Design Document Internal System Services Manual

Revision 0.3

27–April–1988

This manual, which comprises all current Mica system services, was generated directly from the system service source files.

Issued by:

Mark Lucovsky, Bill Muse, Charles Olivier, Lou Perazzoli, and Jim Walker

digital™

Revision History

Date	Revision Number	Author	Summary of Changes
29 FEB 88	0.1	Lucovsky	Initial version.
31 MAR 88	0.2	Lucovsky and others	Second version.
28 APR 88	0.3	Lucovsky and others	Third version.

Contents

1 Object System Services

os$allocate_object

(
IN object_id : e$object_id;
IN allocation_id : e$object_id;
) RETURNS status;

DESCRIPTION

The os$allocate_object service allocates the specified object to the specified allocation object. An allocation object can be a thread, process, job, user, or identifier object.

Each allocation object defines an allocation class. An allocation class is the set of threads that can access an object allocated to an allocation object. If an object is allocated and a thread is a member of the allocation class defined by the allocation object, the thread can access the object (assuming the object access check performed after the allocation check is successful).

The allocation classes defined for each allocation object are:

thread object - The only member of the thread object allocation class is the thread of the thread object that an object is allocated to.

process object - The members of the process object allocation class are the threads of the process object that an object is allocated to and the threads of any child process of the process object that an object is allocated to.

job object - The members of the job object allocation class are the threads of the job object that an object is allocated to.

user object - The members of the user object allocation class are the threads owned by the user who is represented by the user object. An object is allocated to the user object.

identifier object - The members of the identifier object allocation class are the threads that hold the identifier represented by the identifier object.

When an allocation object is deleted, any objects allocated to the object are automatically deallocated.

The visibility of an object determines the allocation objects to which an object can be allocated.

- If the object is at the system level, the object can be allocated to any allocation object.

- If the object is at the job level, the object can be allocated to the job, process, and thread allocation objects.

- If the object is at the process level, the object can be allocated to the process and thread allocation objects.

ARGUMENTS

object_id
Supplies the object id of the object to allocate.

allocation_id

Supplies the object id of the allocation object to which the specified object is allocated.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_invalid_allocation_id	invalid allocation id.
status$_object_type_ mismatch	the object identified by the allocation id is not an allocation object.
status$_object_already_alloc	object is already allocated.
status$_different_alloc_class	the calling thread is not a member of the allocation object's allocation class.
status$_invalid_visibility	the object cannot be allocated because the visibility of the object prevents it from being allocated to the specified allocation object.

os$create_container

(
OUT container_id : e$object_id;
IN object_parameters : e$object_parameters = DEFAULT;
) RETURNS STATUS;

DESCRIPTION

The os$create_container service creates a container. Any type of object except containers and container directories can be inserted into this type of object container.

If the object container id value is specified in the object parameters record, it must identify a container directory. A container can only be inserted into a container directory.

ARGUMENTS

container_id
Returns the object id of the created container.

object_parameters
Supplies the object container in which the object is inserted, the name of the object, and the access control list (ACL) of the object. If this argument is not supplied or if it is supplied but not all values in the object parameter record are supplied, the service applies default values. The default object container is the process container directory, the default name is none, and the default ACL is none.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	the object id of the object container is invalid.
status$_object_type_ mismatch	the object specified by the object container id was not a container directory.
status$_invalid_object	the object to insert is not a container.
status$_duplicate_object	a container having the same type, mode and name was found.
status$_quota_exceeded	the caller does not have enough quota for the specified container or for an expanded container directory.
status$_object_container_full	the container directory is full.

os$create_identifier

(
OUT identifier_id : e$object_id;
IN object_parameters : e$object_parameters;
IN identifier : e$identifier;
) RETURNS status;

DESCRIPTION

The os$create_identifier service creates an identifier object. An identifier object is an allocation object that represents a valid identifier defined on the system. Because it is an allocation object, objects can be allocated to the identifier object. Any thread that is a holder of the identifier represented by the identifier object can access any objects allocated to the identifier object.

To create an identifier object, the caller must hold the identifier that the identifier object is to represent.

The identifier object is inserted in the exec$identifier_container system level container. The name of the object is the alphanumeric name of the identifier the object represents.

ARGUMENTS

identifier_id
Returns the object id of the created identifier object.

object_parameters
Supplies the object container in which the object is inserted, the name of the object, and the access control list (ACL) of the object. The values for the name and object container are ignored. If a value for the ACL is not supplied, the default is

None.

identifier - Supplies the identifier that the identifier object represents.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_identifier	the caller is not a holder of the specified identifier.
status$_duplicate_object	duplicate object found in object container.

os$create_reference_id

(
IN object_id : e$object_id;
IN container_id : e$object_id = DEFAULT;
OUT reference_id : e$object_id;
) RETURNS status;

DESCRIPTION

The os$create_reference_id service creates a reference id to an object. A reference id ensures that as long as the reference id exists, the object cannot be deleted.

A reference id can only be created for objects whose principal id still exists.

The container through which the reference id identifies the object must be at a less visible level than the principal object id's container.

A reference id cannot be created for an object that does not allow reference ids. For example, container directories and containers do not allow reference ids.

ARGUMENTS

object_id
Supplies the object id of the object that a reference id is created for.

container_id
Supplies the container id of the container thru which the object is referenced.

reference_id
Returns the reference id.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_invalid_container_id	invalid container id.
status$_object_type_ mismatch	the object type of the specified container was not a container.
status$_reference_not_ allowed	the object does not allow reference ids.
status$_invalid_target_level	the level of the container is not more visible than the object's container.

os$deallocate_object

(
IN object_id : e$object_id;
) RETURNS STATUS;

DESCRIPTION

The os$deallocate_object service deallocates the specified object.

The caller must be a member of the allocation object's allocation class in order to deallocate the object.

ARGUMENTS

object_id
Supplies the object id of the object to deallocate.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_object_not_allocated	object not allocated.

os$delete_object_id

(
IN object_id : e$object_id;
) RETURNS STATUS;

DESCRIPTION

The os$delete_object_id service deletes the object id of the specified object. When all object ids that identify the object have been deleted, the object is no longer accessible.

Paged or nonpaged pool quota is returned to the correct level when the object id is deleted. If the object identified by the deleted object id was at the system level, no quota is returned.

If the object id count decrements to 0, the remove object service routine specified by the object's OTD is called. After the remove object service routine returns, this service dereferences the object by calling obj$dereference_object.

ARGUMENTS

object_id
Supplies the object id to delete.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.

os$delete_object_name

(
IN object_id : e$object_id;
) RETURNS status;

DESCRIPTION

The os$delete_object_name service deletes the specified object's name and removes the name from the object container's object name table.

ARGUMENTS

object_id
Supplies the object id of the object whose name is deleted.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_name_already_ deleted	the object name of the object was already deleted.

os$get_objcon_information

(
IN object_container_id : e$object_id;
IN item_list : POINTER e$item_list_type;
) RETURNS status;

DESCRIPTION

The os$get_objcon_information service returns the object ids of objects in the object container and the logical names in the object containers' logical name table. An object container is either a container directory or container.

Object ids are returned in the e$c_object_id_list item. This item is of type e$object_id_list. The e$object_id_list type is made up of the following fields:

- length - This field is set by the caller and indicates to the service the number of entries in the object_id field.

- last_valid_entry - This field is set by the service and indicates to the caller the last entry in the object_id field that contains a valid value.

- context - This field maintains context across multiple calls to the service. It is set by the caller and the service.

- object_id - This field is set by the service and indicates to the caller the object ids that identify objects in the object container.

As described above, the last_valid_entry field indicates the last entry in the object_id field that contains a valid value. This field can have the following values:

- If the value of this field is zero, the service did not return any object ids. This means the object container does not hold any objects. A subsequent call to the service would not return additional object ids.

- If the value is non-zero and is less than the maximum number of entries, the service returned the object ids that identify all the objects in the object container. A subsequent call to the service would not return additional object ids.

- If the value is non-zero and is equal to the maximum number of entries, the service may have returned the object ids that identify all the objects in the object container. The caller must examine the status returned by the service to determine if all the object ids were returned. If the status returned was status$_no_more_info, the service returned all the object ids and a subsequent call to the service would not return additional object ids. If the status returned was status$_normal, the service did not return all the object ids and a subsequent call to the service might return additional object ids.

Note that the service might return additional object ids. At the time the call completed, the service may have found more objects and therefore more object ids than could be returned. Between the time the first call completes and a subsequent call is made, the objects could be deleted. The subsequent call would then return a status of status$_no_more_info and the last_valid_entry field would have a value of zero.

As described above, the context field maintains context across multiple calls to the service. The context field can have the following values:

- zero - When the context field is zero, the service attempts to set entries in the object_id field beginning with the object id of the first object found in the object container.

- nonzero - When the context field is nonzero, the service attempts to set entries in the object_id field beginning with the object id of the next object found in the object container.

For the initial call, the caller sets the value of the context field to 0. For subsequent calls when additional object ids can be returned, the caller should not modify the value of the context field.

Logical names are returned in the e$c_logical_name_list item. This item is of type e$logical_name_list. The e$logical_name_list type is made up of the following fields:

- length - This field is set by the caller and indicates to the service the number of entries in the logical_name field.

- last_valid_entry - This field is set by the service and indicates to the caller the last entry in the logical_name field that contains a valid value.

- context - This field maintains context across multiple calls to the service. It is set by the caller and the service.

- logical_name - This field is set by the service and indicates to the caller the logical names in the object container's logical name table.

The use of the last_valid_entry and the context fields is similar as described for the object id list and is not described.

Note that the caller can request object ids and logical names in the same item list. If more information can be returned for either the object id list or the logical name list, the status returned is status$_normal. If no more information can be returned for either list, the status returned is status$_no_more_info. In both cases, the caller should examine the last_valid_entry in each list to determine the number of entries, if any, were returned.

ARGUMENTS

object_container_id
Supplies the object id of the object container for which information is returned. The object id identifies either a container directory or a container.

item_list
Supplies the item list identifying the information the service should return.

code	pointer type	action
e$c_object_id_list	e$object_id_list	Returns a list of object ids that identify the objects in the object container.
e$c_logical_name_list	e$logical_name_list	Returns a list of logical names contained in the object container's logical name table.

RETURN VALUES

status$_normal	normal, successful completion. The object container was found and some of the object ids or logical names were returned. A subsequent call to this service may return additional information.
status$_no_more_info	normal, successful completion. The object container was found and all of the object ids or logical names were returned. A subsequent call to this service will not return additional information.
status$_invalid_object_id	invalid object id.
status$_object_type_ mismatch	the object type of the specified object container was not a container directory or container.

os$get_object_information

(
IN object_id : e$object_id;
IN item_list : POINTER e$item_list_type;
) RETURNS status;

DESCRIPTION

The os$get_object_information service returns information about the specified object. The information is control information about the object and is general for all objects.

ARGUMENTS

object_id
Supplies the object id of the object for which information is returned.

item_list
Supplies the item list identifying the information the service should return.

code	pointer type	action
e$c_pointer_count	integer	Returns the number of outstanding pointers to the object.
e$c_object_id_count	integer	Returns the number of object ids that identify the object.
e$c_level	e$level	Returns the level of visibility of the object. The level can be e$c_process_level, e$c_job_level, or e$c_system_level.
e$c_object_type_name	string	Returns the object type name of the object.
e$c_otd_id	e$object_id	Returns the object id of the object's OTD.
e$c_object_container_id	e$object_id	Returns the object id of the object's object container. This object id identifies either a container directory or a container. This field is valid only if the object's principal id has not been deleted. See e$c_object_state.
e$c_principal_object_id	e$object_id	Returns the object id of the object's principal id. This field is valid only if the object's principal id has not been deleted. See e$c_object_state.
e$c_nonpaged_pool_charge	integer	Returns the amount of nonpaged pool charged when the object was inserted into its object container.
e$c_paged_pool_charge	integer	Returns the amount of paged pool charged when the object was inserted into its object container.
e$c_name	varying_string	Returns the object's name. This field is valid only if the object's principal id has not been deleted. See e$c_object_state.

code	pointer type	action
e$c_owner	e$identifier	Returns the object's owner.
e$c_acl	e$access_control_list	Returns the object's access control list.
e$c_allocation_object_id	e$object_id	Returns the object id of the object's allocation object. This field is valid only if the object is allocated. See e$c_object_state.
e$c_mode	k$processor_mode	Returns the processor mode of the object. The mode of the object can be k$c_user or k$c_kernel.
e$c_object_state	set of e$object_state	Returns information about the current state of the object. The states are: e$c_transfer_inhibit — the object cannot be transferred. e$c_reference_inhibit — reference ids cannot be created to identify the object. e$c_temporary — the object has been marked as temporary. e$c_dispatcher_object — the object has a kernel dispatcher object. This allows the object to be waited on. e$c_allocated — the object is allocated. e$c_principal_id_deleted — the principal id of the object has been deleted. e$c_transferred — the object has been transferred.
e$c_oid_object_container_id	e$object_id	Returns the object id of the object container through which the object is identified by the specified object id.
e$c_oid_level	e$level	Returns the level of visibility of the object when identified by the specified object id. The level can be e$c_process_level, e$c_job_level, or e$c_system_level.
e$c_oid_object_id_type	e$object_id_type	Returns the type of object id. The type of id can be e$c_principal_id or e$c_reference_id.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.

os$get_otd_information

(
IN otd_id : e$object_id;
IN item_list : POINTER e$item_list_type;
) RETURNS status;

DESCRIPTION

The os$get_otd_information service returns information about the specified object.

ARGUMENTS

otd_id
Supplies the object id of the otd object for which information is returned.

item_list
Supplies the item list identifying the information the service should return.

code	pointer type	action
e$c_object_type_name	string	Returns the name of the object type described by the OTD.
e$c_object_count	integer	Returns the count of the number of objects of this type.
e$c_waitable	boolean	Returns a value of true if objects of the type described by the OTD can be waited on. Returns a value of false if objects cannot be waited on.
e$c_create_disable	boolean	Returns the state of the create disable flag. If the value is false, objects of this type can be created. If the value is true, objects of this type cannot be created.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_object_type_mismatch	the object type of the specified object was not an otd.

os$mark_temporary

(
IN object_id : e$object_id;
) RETURNS status;

DESCRIPTION

The os$mark_temporary service marks the specified object as temporary.

This service is used to cause the principal id of an object to be deleted when all reference ids to the object have been deleted. If the principal id has already been deleted, the last deleted reference id causes the object to be deleted.

Only job and system level objects can be marked as temporary.

Container directories and containers cannot be marked as temporary.

ARGUMENTS

object_id
Supplies the object id of the object to mark as temporary.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_invalid_object_level	the object is a process level object.
status$_already_temporary	the object is already temporary.
status$_temporary_not_allowed	the object cannot be marked as temporary.

os$set_object_name

(
IN object_id : e$object_id;
IN name : string (*);
) RETURNS status;

DESCRIPTION

The os$set_object_name service sets the specified object's name and inserts the name in the object's object container object name table.

The name of an object can be set only if the principal id of the object exists.

ARGUMENTS

object_id
Supplies the object id of the object whose name is set.

name
Supples the name that the object name's name is set to.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_duplicate_object	object found having the same mode, type, and name.

os$transfer_mark_temporary

(
IN container_id : e$object_id;
IN delete : boolean = false;
IN OUT object_id : e$object_id;
) RETURNS status;

DESCRIPTION

The os$transfer_mark_temporary service transfers the object along with its name to a more visible container and marks the object as temporary.

When an object is transferred to the target container, it is possible that an object already exists having the same name, object type, and mode. If a duplicate object does exit, the caller can specify the action to perform. If the action is not to delete the object specified by the caller, the service does not transfer the object and returns an error status. Note that the object id is unchanged. If the action is to delete the object, the service creates a reference id to the already existing object, deletes the object id of the object specified by the caller, and returns the reference id to the caller. The reference id is returned via the object_id parameter.

If a duplicate object does not exist, the service transfers the object to the target container, creates a reference id to the object, and returns the reference id to the caller. The reference id is returned via the object_id parameter.

The object cannot be transferred if any one of the following conditions are true:

- the object has reference ids. This means that the object id specified by the object_id parameter is the principal id of the object. - the object is not allowed to be transferred. - an object having the same name, type, and mode already exists in the target container and the delete action was specified as false.

Container directories and containers cannot be transferred and marked as temporary.

ARGUMENTS

container_id
Supplies the object id of the container into which the object is transferred.

delete
Supplies the action to perform if a duplicate object is found in the container. If the value is false, the service does not transfer the specified object and returns an error status. If the value is true, the service creates a reference id to the already existing object, deletes the object specified by the caller, and returns the reference id to the caller. If a value is not specified, a value of false is assumed.

object_id

Supplies the object id of the object that is transfered and marked temporary. This object id must be the object's principal id. Returns the reference id of the temporary object.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_invalid_container_id	invalid container id.
status$_object_type_mismatch	the object type of the specified container was not a container.
status$_object_already_temp	the object is already temporary.
status$_temporary_not_allowed	the object cannot be marked as temporary.
status$_duplicate_temporary	a duplicate object exists in the target container and is temporary.
status$_duplicate_not_temporary	a duplicate object exists in the target container and is not temporary.
status$_invalid_target_level	the level of the target container is not more visible than the original container.
status$_object_reference_ids	the object id has reference ids.
status$_invalid_object_id_count	the object id count of the specified object is not 1.

os$translate_object_name

(
IN object_container_id : e$object_id = DEFAULT;
IN name : string (*);
IN object_type_name : string (*);
IN case_sensitive : boolean = true;
OUT object_id : e$object_id;
) RETURNS status;

DESCRIPTION

The os$translate_object_name service searches the specified object container for an object having the specified object name and object type name. If an object is found, the service returns the object id of the object. The object id is used as input to other services to identify the object that the service is to operate on.

The service locates the object name using one of two search methods as specified by the case_sensitive parameter. If the value is false, the service performs a case blind search. If the value is true, the service performs a case sensitive search.

A case blind search locates the first object name whose uppercase representation matches the uppercase representation of the object name specified by the caller. Multiple object names in the object container may match but only the first object name found is matched.

A case sensitive search locates the object name whose name exactly matches the object name specified by the caller. Only one object name can match.

The service matches the object type name using a case sensitive search.

The caller can optionally specify the object container parameter. If the parameter is not specified, the service searches the object name tables of the process, job, and system container directories. If a match is found, the object id that identifies the object is returned to the caller. If the parameter is specified, the service searches the object name table of the specified object container.

If the previous mode of the caller is user, the service tries to match a user mode object having the specified name and object type name in the target object container. If a name is found, the object id of the user mode object is returned to the caller. If a name is not found, the service tries to match a kernel mode object with the same search criteria. If a name is found, the object id of the kernel mode object is returned to the caller.

ARGUMENTS

object_container_id
Supplies the name of the object container whose object name table is searched. The object id identifies either a container directory or a container.

name
Supplies the name of the object to find.

object_type_name
Supplies the object type name of the object to find.

case_sensitive
Supplies the search method used to locate the object name. A value of false indicates a case blind search. A value of true indicates a case sensitive search.

object_id
Returns the object id of the matching object.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_name_length	length of the object name or object type name was not valid.
status$_invalid_object_type	invalid object type specified by the object type name.
status$_invalid_object_id	the object id of the object container is invalid.
status$_object_type_mismatch	the object specified by the object container id was not a container directory or a container.
status$_object_name_not_found	object name not found.

2 Logical Name System Services

os$create_logical_name

(
IN object_container_id : e$object_id;
IN logical_name : string (*);
IN supersede : boolean = true;
IN logical_name_attributes : SET e$lognam_attributes [..] = [];
IN OUT equivalence_name_list : e$equivalence_name_list;
) RETURNS status;

DESCRIPTION

The os$create_logical_name service creates the specified logical name in the specified object container.

Before the service creates the logical name, it performs a case sensitive search for the logical name in the object container. If a logical name is not found, the service creates the logical name. If a logical name is found, the service takes the action specified by the supersede parameter. If a value of false is specified, the logical name specified by the caller is not created and the service fails. If a value of true is specified, the logical name that was found is deleted and the logical name specified by the caller is created.

Logical names and equivalence names contain 1-255 characters. The characters that form the name can be any character in the character set.

A logical name can have 1-128 equivalence names.

Equivalence names are specified in the equivalence_name_list parameter. This parameter is of type e$equivalence_name_list. The e$equivalence_ name_list type is made up of the following fields:

- length - This field is set by the caller and indicates to the service the number of entries in the equivalence_name field.

- last_valid_entry - This field is set by the caller and indicates to the service how many valid entries are in the equivalence_name field.

- context - This field is set by the service when an entry in the equivalence_ name field is invalid. The context field indicates to the caller the entry that is invalid.

- equivalence_name - This field is set by the caller and indicates to the service the equivalence name or names to assocaiate with the specified logical name.

A logical name can have attributes associated with it. An attribute denotes a characteristic of the logical name. The following logical name attributes are defined:

- confine - The confine attribute indicates that the logical name should not be transferred when an object container is transferred. If the logical name has the confine attribute, the object container transfer service deletes the logical name as the transfer is performed. The caller gives the logical name the confine attribute by setting e$c_confine_lognam_attr in the logical_name_attributes parameter. If the confine attribute is not given to the logical name, the logical name is transferred.

- noalias - The noalias attribute indicates to os$create_logical_name that the logical name cannot be duplicated in the object container at an outer access mode. If another logical name with the same name already exists in the object container at an outer access mode and the caller of os$create_logical_name specifies the noalias attribute, os$create_logical_name first deletes the logical name at the outer access mode and then creates the logical name at the inner access mode. The caller gives the logical name the noalias attribute by setting e$c_noalias_lognam_attr in the logical_name_attributes parameter. If the noalias attribute is not given to the logical name, the logical name can have a logical name with the same name at an outer access mode.

- noshow - The noshow attribute indicates to the caller of os$translate_logical_name that the logical name should not be displayed. General show logical name utilities examine this attribute to determine if the logical name should be displayed. The caller gives the logical name the noshow attribute by setting e$c_noshow_lognam_attr in the logical_name_attributes parameter. If the noshow attribute is not given to the logical name, the logical name can be displayed.

Each entry in the equivalence name list specifies an equivalence name and the attributes to give to the equivalence name. An attribute denotes a characteristic of the equivalence name. The following equivalence name attributes are defined:

- concealed - The concealed attribute indicates to the caller of os$translate_logical_name that the equivalence name should not be displayed. General show logical name utilities examine this attribute to determine if the equivalence name should be displayed. The caller gives the equivalence name the concealed attribute by setting the e$c_concealed_eqvnam_attr in the attributes field of the equivalence name entry. If the concealed attribute is not given to the equivalence name, the equivalence name can be displayed.

- terminal - The terminal attribute indicates to the caller of os$translate_logical_name that the equivalence name should not be translated as if it were a logical name. The caller gives the equivalence name the terminal attribute by setting the e$c_terminal_eqvnam_attr in the attributes field of the equivalence name entry. If the terminal attribute is not given to the equivalence name, the equivalence name can be translated as if it were a logical name.

ARGUMENTS

object_container_id
Supplies the object id of the object container whose logical name table the logical name is created in. The object id identifies either a container directory or a container.

logical_name
Supplies the name of the logical name to create. The size of the name can be 1 to 255 characters. Any character can be used in the logical name.

supersede
Supplies the action to perform if a matching logical name is found in the object container's logical name table.

logical_name_attributes
Supplies a set containing the attributes of the logical name.

equivalence_name_list

Supplies the equivalence names associated with the logical name. Returns in the context field the number of the entry that is invalid. If all entries are valid, the value of the context field is 0.

RETURN VALUES

status$_normal	normal, successful completion. The logical name was created.
status$_logical_name_ superseded	normal, successful completion. The logical name was created and a previously existing logical name with the same name was deleted.
status$_invalid_object_id	invalid object container id.
status$_object_type_ mismatch	the object type of the specified object container was not a container directory or container.
status$_invalid_name_length	length of the logical name or the equivalence name was not valid.
status$_invalid_eqv_name_ count	the count of the number of equivalence names was invalid.
status$_duplicate_logical_ name	duplicate logical name was found.
status$_quota_exceeded	quota was exceeded while trying to create the logical name.

os$delete_logical_name

(
IN object_container_id : e$object_id;
IN logical_name : string (*);
) RETURNS status;

DESCRIPTION

The os$delete_logical_name service deletes the specified logical name from the specified object container.

The service performs a case sensitive search for the logical name in the object container.

ARGUMENTS

object_container_id
Supplies the object id of the object container whose logical name table is searched. The object id identifies either a container directory or a container.

logical_name
Supplies the logical name to delete.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object container id.
status$_object_type_mismatch	the object type of the specified object container was not a container directory or container.
status$_invalid_name_length	length of the logical name was not valid.
status$_logical_name_not_found	logical name was not found.

os$translate_logical_name

(
IN object_container_id : e$object_id;
IN logical_name : string (*);
IN case_sensitive : boolean = true;
IN OUT equivalence_name_list : e$equivalence_name_list;
OUT logical_name_attributes : SET e$lognam_attributes [..] OPTIONAL;
) RETURNS status;

DESCRIPTION

The os$translate_logical_name service searches the specified object container for the specified logical name. If the logical name is found, the service returns the logical name's equivalence names.

The service locates the logical name in the object container using one of two search methods as specified by the case_sensitive parameter. If the value is false, the service performs a case blind search. If the value is true, the service performs a case sensitive search.

A case blind search locates the first logical name whose uppercase representation matches the uppercase representation of the logical name specified by the caller. Multiple logical names in the object container may match but only the first logical name found is matched.

A case sensitive search locates the logical name whose name exactly matches the logical name specified by the caller. Only one logical name in the object container can match.

Equivalence names are returned in the equivalence_name_list parameter. This parameter is of type e$equivalence_name_list. The e$equivalence_name_list type is made up of the following fields:

- length - This field is set by the caller and indicates to the service the number of entries in the equivalence_name field.

- last_valid_entry - This field is set by the service and indicates to the caller the last entry in the equivalence_name field that contains a valid value.

- context - This field maintains context across multiple calls to the service. It is set by the caller and the service.

- equivalence_name - This field is set by the service and indicates to the caller the equivalence name or names assocaiated with the logical name.

As described above, the last_valid_entry field indicates the last entry in the equivalence_name field that contains a valid value. This field can have the following values:

- If the value of this field is zero, the service did not return any equivalence names associated with the logical name. A subsequent call to the service would not return additional equivalence names.

- If the value is non-zero and is less than the maximum number of entries, the service returned all the equivalence names associated with the logical name. A subsequent call to the service would not return additional equivalence names.

- If the value is non-zero and is equal to the maximum number of entries, the service may have returned all the equivalence names associated with the logical name. The caller must examine the status returned by the service to determine if all the equivalence names were returned. If the status returned was status$_no_more_info, the service returned all the equivalence names and a subsequent call to the service would not return additional equivalence names. If the status returned was status$_normal, the service did not return all the equivalence names and a subsequent call to the service would return additional equivalence names.

As described above, the context field maintains context across multiple calls to the service. The context field can have the following values:

- zero - When the context field is zero, the service attempts to set entries in the equivalence_name field beginning with the first equivalence name associated with the logical name.

- nonzero - When the context field is nonzero, the service attempts to set entries in the equivalence_name field beginning with the next equivalence name associated with the logical name indicated by the value in the context field.

For the initial call, the caller sets the value of the context field to 0. For subsequent calls when additional equivalence names can be returned, the caller should not modify the value of the context field.

Note, if multiple calls to the service are required to return all the equivalence names, the logical name may be deleted in between the calls.

ARGUMENTS

object_container_id
Supplies the object id of the object container whose logical name table is searched. The object id identifies either a container directory or a container.

logical_name
Supplies the name of the logical name to translate.

case_sensitive
Supplies the search method used to locate the logical name. A value of false indicates a case blind search. A value of true indicates a case sensitive search.

equivalence_name_list
Supplies (in the length field) the number of entries in the equivalence name field. Supplies (in the context field) the context of the service. Returns (in the last_valid_entry field) the last entry in the equivalence_ name field that contains a valid value. Returns (in the context field) the context for the next call to the service. Returns (in the equivalence_name field) some or all of the equivalence names associated with the logical name.

logical_name_attributes

Returns a set containing the attributes of the logical name. See os$create_logical_name for an explanation of the logical name attributes.

RETURN VALUES

status$_normal	normal, successful completion. The logical name was found and some of the equivalence names were returned. A subsequent call to this service may return additional information.
status$_no_more_info	normal, successful completion. The logical name was found and all of the equivalence names were returned. A subsequent call to this service will not return additional information.
status$_invalid_object_id	invalid object container id.
status$_object_type_mismatch	the object type of the specified object container was not a container directory or container.
status$_invalid_name_length	length of the logical name was not valid.
status$_logical_name_not_found	logical name was not found.

3 Wait System Services

os$wait_multiple

(
IN OUT object_id_list : e$object_id_list;
IN time_out : large_integer OPTIONAL;
IN wait_type : e$wait_type = e$c_wait_any;
OUT object_number : integer;
) RETURNS return_status : status;

DESCRIPTION

The os$wait_multiple service suspends the execution of the caller until one or all of the specified objects become signalled or the specified time interval expires.

The object ids that identify the objects to wait on are specified in the object_id_list parameter. This parameter is of type e$object_id_list. The e$object_id_list type is made up of the following fields:

- length - This field is set by the caller and indicates to the service the number of entries in the object_id field.

- last_valid_entry - This field is set by the caller and indicates to the service how many valid entries are in the object_id field.

- context - This field is set by the service when an entry in the object_id field is invalid. The context field indicates to the caller the entry that is invalid.

- object_id - This field is set by the caller and indicates to the service the object ids that identify the objects to wait on.

ARGUMENTS

object_id_list
Supplies the object ids that identify the objects to wait on. Returns in the context field the number of the entry that is invalid. If all entries are valid, the context is 0.

time_out
The amount of time in 100 nanosecond units that can expire before the wait is timed out.

wait_type
Supplies the type of wait. If e$c_wait_any is specified, any object in the object list that is signalled satisfies the wait. If e$c_wait_all is specified, all objects in the object list must be signalled to satisfy the wait. If a value is not specified, e$c_wait_any is assumed.

object_number
Returns the number of the object in the object id list that satisfied the wait. If the wait times out, the object number is 0.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_invalid_object_count	the count of the number of objects to wait on was invalid.
status$_wait_not_supported	wait not supported by the specified object.
status$_wait_timeout	wait was not satisfied before the time out period.

os$wait_single

(
IN object_id : e$object_id;
IN time_out : large_integer OPTIONAL;
) RETURNS return_status : status;

DESCRIPTION

The os$wait_single service suspends the execution of the caller until the specified object becomes signalled or the specified time interval expires.

ARGUMENTS

object_id
Supplies the object id that identifies the object to wait on.

time_out
The amount of time in 100 nanosecond units that can expire before the wait is timed out.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_object_type_ mismatch	object type specified does not match the object type of the object.
status$_wait_not_supported	wait not supported by the specified object.
status$_wait_timeout	wait was not satisfied before the time out period.

4 Event System Services

os$clear_event

(
IN event_id : e$object_id;
OUT previous_state : boolean;
) RETURNS return_status : status;

DESCRIPTION

The os$clear_event service clears the state of the specified event to not signalled.

ARGUMENTS

event_id
Supplies the object id of the event to clear.

previous_state
Returns the previous state of the event. A value of false indicates that the state of the event was clear (not signalled). A value of true indicates that the state of the event was set (signalled).

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_object_type_ mismatch	object type specified does not match the object type of the object.

os$create_event

(
OUT event_id : e$object_id;
IN object_parameters : e$object_parameters = DEFAULT;
IN autoclear_flag : boolean = false;
IN initial_state : boolean = false;
) RETURNS return_status : status;

DESCRIPTION

The os$create_event service creates an event object.

An event can have two states: clear and set. When an event is clear it is not signalled. When an event is set it is signalled. Only an event that has been signalled satisfies a wait. An event is signalled by calling os$set_event.

The creator of an event can specify that the event is automatically cleared when the event satisfies a wait. If multiple threads are waiting on the event, only the first thread's wait is satisfied; the remaining threads must wait until the event is set again. If the object is created without automatic clearing, the event remains set until explicitly cleared. If multiple threads are waiting on the event, all the waits are satisfied. An event is cleared by calling os$clear_event.

ARGUMENTS

event_id
Returns the object id of the created event.

object_parameters
Supplies the object container in which the object is inserted, the name of the object, and the access control list (ACL) of the object. If this argument is not supplied or if it is supplied but not all values in the object parameter record are supplied, the service applies default values. The default object container is the process private container, the default name is none, and the default ACL is none.

autoclear_flag
Supplies the action taken when a wait on the event is satisfied. If the value is false, the state of the event is not changed; otherwise, the state is cleared. If this argument is not supplied, the state is not changed.

initial_state
Supplies the initial state of the event. If the value is false, the initial state is cleared (not signalled); otherwise, it is set (signalled). If this argument is not supplied, the state is cleared.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_object_type_mismatch	object type specified does not match the object type of the object.
status$_invalid_object	invalid object.
status$_duplicate_object	duplicate object found in object container.
status$_object_container_full	object container full.

os$pulse_event

(
IN event_id : e$object_id;
OUT previous_state : boolean;
) RETURNS return_status : status;

DESCRIPTION

The os$pulse_event service sets the state of the specified event to signalled, services all the threads waiting on the event, and clears the state of the specified event to not signalled.

The service ignores the autoclear flag that was specified when the event was created

ARGUMENTS

event_id
Supplies the object id of the event to clear.

previous_state
Returns the previous state of the event. A value of false indicates that the state of the event was clear (not signalled). A value of true indicates that the state of the event was set (signalled).

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_object_type_ mismatch	object type specified does not match the object type of the object.

os$read_event

(
IN event_id : e$object_id;
OUT state : boolean;
) RETURNS return_status : status;

DESCRIPTION

The os$read_event service reads the state of the specified event.

ARGUMENTS

event_id
Supplies the object id of the event to read.

state
Returns the current state of the event. A value of false indicates that the state of the event is clear (not signalled). A value of true indicates that the state of the event is set (signalled).

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_object_type_ mismatch	object type specified does not match the object type of the object.

os$set_event

(
IN event_id : e$object_id;
OUT previous_state : boolean;
) RETURNS return_status : status;

DESCRIPTION

The os$set_event service sets the state of the specified event to signalled.

ARGUMENTS

event_id
Supplies the object id of the event to set.

previous_state
Returns the previous state of the event. A value of false indicates that the state of the event was clear (not signalled). A value of true indicates that the state of the event was set (signalled).

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_object_type_mismatch	object type specified does not match the object type of the object.

5 Semaphore System Services

os$create_semaphore

(
OUT semaphore_id : e$object_id;
IN object_parameters : e$object_parameters;
IN initial_count : integer;
IN maximum_count : integer;
) RETURNS status;

DESCRIPTION

This os$create_semaphore service creates a semaphore object.

(The following description is brought to you by the Kernel.) A semaphore object is used to control access to a resource but not necessarily in a mutually exclusive fashion. A semaphore acts as a gate through which a variable number of threads can pass concurrently, up to a specified limit. The gate is open (signaled state) as long as there are resources available. When the number of resources that may be concurrently in use has been exhausted, the gate is closed (not-signaled state). The gating mechanism of a semaphore is implemented by a counter. Waiting on a semaphore waits until a resource is available and decrements the count. Releasing the semaphore increments the count and allows another thread to pass through the gate.

ARGUMENTS

semaphore_id
Returns the object id of the created semaphore.

object_parameters
Supplies the object container in which the object is inserted, the name of the object, and the access control list (ACL) of the object. If this argument is not supplied or if it is supplied but not all values in the object parameter record are supplied, the service applies default values. The default object container is the process private container, the default name is none, and the default ACL is none.

initial_count
Supplies the initial count of the semaphore. The intitial count must be less than or equal to the maximum count.

maximum_count
Supplies the maximum count the semaphore can attain. The maximum count must be greater than zero.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_object_type_ mismatch	object type specified does not match the object type of the object.
status$_duplicate_object	duplicate object found in object container.
status$_object_container_full	object container full.
status$_invalid_initial_count	the value specified as the initial count was greater than the maximum.
status$_invalid_maximum_ count	the value specified as the maximum count was not greater than zero.

os$read_semaphore

```
(
IN semaphore_id : e$object_id;
OUT count : integer;
) RETURNS status;
```

DESCRIPTION

The os$read_semaphore service reads the count of the specified semaphore.

ARGUMENTS

semaphore_id
Supplies the object id of the semaphore object to read.

count
Returns the count of the semaphore.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_object_type_ mismatch	object type specified does not match the object type of the object.

os$release_semaphore

(
IN semaphore_id : e$object_id;
IN release_count : integer = 1;
OUT previous_count : integer;
) RETURNS status;

DESCRIPTION

The os$release_semaphore service releases the specified semaphore. This action causes the semaphore count to be incremented by the specified count. If the count was 0 before it was incremented, the the state of the semaphore is set to signaled.

The release_count argument specifies the value that is added to the semaphore count. If a value for this argument is not specified, the semaphore count is incremented by 1. The resulting semaphore count must not exceed the maximum count of the semaphore.

ARGUMENTS

semaphore_id
Supplies the object id of the semaphore object to release.

release_count
Supplies the value that is added to the semaphore count.

previous_count
Returns the count of the semaphore before the count was incremented.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_object_type_ mismatch	object type specified does not match the object type of the object.
status$_invalid_release	the release of the semaphore caused the the count to exceed the maximum count.

6 Interval System Services

os$cancel_timer

(
IN timer_id : e$object_id;
OUT timer_state : boolean;
) RETURNS status;

DESCRIPTION

Cancels a timer object. If a timer object has been set with an AST, only the thread that originally set the timer may cancel it.

ARGUMENTS

timer_id
supplies the object id of the timer object

timer_state
returns true if the timer was currently active, false otherwise

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_invalid_cancel_timer	the calling thread is not the thread that set the timer with an AST
others	object id translation errors

os$create_timer

(
OUT timer_id : e$object_id;
IN object_parameters : e$object_parameters = DEFAULT;
) RETURNS status;

DESCRIPTION Creates and initializes a timer object. The default object container is process private

ARGUMENTS

timer_id
returns the object id of the resulting timer object

object_parameters
supplies the object type independent parameters governing the creation of the timer object

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_duplicate_object	a timer with the same name already exists in the specified container
others	object id translation errors

os$read_timer

(
IN timer_id : e$object_id;
OUT timer_state : boolean;
) RETURNS status;

DESCRIPTION reads the signaled state of a timer object

ARGUMENTS

timer_id
supplies the object id of the timer object

timer_state
returns true if the timer is in the signaled state , false otherwise

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
others	object id translation errors

os$set_timer

(
IN timer_id : e$object_id;
IN due_time : large_integer;
IN ast_procedure : k$normal_ast_routine = NIL;
IN ast_parameter : POINTER anytype CONFORM = NIL;
) RETURNS status;

DESCRIPTION

Sets a timer to expire in due_time. Timers are waitable objects. Waits are satisfied when the timer expires.

When timers are used with ASTs, the system_value parameter is the current system time in absolute UTC.

ARGUMENTS

timer_id
supplies the object id of the timer to set

due_time
supplies the number of 100ns units of time that should elapse before the timer expires if due_time is negative, the timer is "relative", or the timer will expire (-due_time) units of time after the set timer call is made. Positive values of due_time implys absolute time in UTC.

ast_procedure
supplies the procedure that should be called when the timer expires. If defaulted, no procedure is called. If the previous mode is k$c_user, then the procedure is called as a user mode ast procedure, otherwise, it is called as a kernel mode ast procedure.

ast_parameter
supplies the context passed to the ast procedure. If the ast procedure is defaulted, then this parameter is ignored.

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_invalid_cancel_timer	the timer is set with an AST, and the calling thread is not the thread that originally set the timer with an AST
others	object id translation errors

7 Process System Services

os$create_exit_handler_process

(
IN handler_procedure : k$normal_ast_routine;
IN handler_context : POINTER anytype CONFORM = NIL;
IN handler_placement : e$exit_handler_placement = e$c_beginning_of_list;
OUT handler_id : e$exit_handler_id;
) RETURNS status;

DESCRIPTION

This service is used to create a process level exit handler. Exit handlers are called as user mode AST routines during exit. Process level exit handlers are processed when a the last thread in a process calls os$exit_thread( ), and after all of the thread level exit handlers have been processed. The exit handler list head stored in the exiting threads PCR is processed in order. Each handler found in the list is removed and then called as an AST routine. This interface supports placement of an exit handler at either the beginning or end of the exit handler list head. Placement is under the control of the handler_placement parameter which defaults to beginning of the list. Once created, a handler is assigned a handler_id. This return value may be used to delete an existing exit handler.

ARGUMENTS

handler_procedure
Supplies the exit handler procedure to be executed when this handler is processed

handler_context
Supplies a parameter to be passed to the handler_procedure when the handler is processed.

handler_placement
Supplies exit handler placement control.

handler_id
Returns the handler ID of the exit handler. This argument is only valid if the service returns with status$_normal.

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessible
status$_not_supported	an attempt to call this service from a system thread was made, or the service was called after kernel mode exit processing has started.

os$create_exit_handler_thread

(
IN handler_procedure : k$normal_ast_routine;
IN handler_context : POINTER anytype CONFORM = NIL;
IN handler_placement : e$exit_handler_placement = e$c_beginning_of_list;
OUT handler_id : e$exit_handler_id;
) RETURNS status;

DESCRIPTION

This service is used to create a thread level exit handler. Exit handlers are called as user mode AST routines during exit. Thread level exit handlers are processed when a thread calls os$exit_thread( ). The exit handler list head stored in the exiting threads TCR is processed in order. Each handler found in the list is removed and then called as an AST routine. This interface supports placement of an exit handler at either the beginning or end of the exit handler list head. Placement is under the control of the handler_placement parameter which defaults to beginning of the list. Once created, a handler is assigned a handler_id. This return value may be used to delete an existing exit handler.

ARGUMENTS

handler_procedure
Supplies the exit handler procedure to be executed when this handler is processed

handler_context
Supplies a parameter to be passed to the handler_procedure when the handler is processed.

handler_placement
Supplies exit handler placement control.

handler_id
Returns the handler ID of the created exit handler. This argument is only valid if the service returns with status$_normal.

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessible
status$_not_supported	an attempt to call this service from a system thread was made, or the service was called after kernel mode exit processing has started.

os$create_exit_status

(
OUT exit_status_id : e$object_id;
IN object_parameters : e$object_parameters = DEFAULT;
) RETURNS status;

DESCRIPTION

Create and initialize an exit status object. If the container id stored in object parameters is defaulted, then process private is assumed.

ARGUMENTS

exit_status_id
object id of created exit status object

object_parameters
the object type independant parameters of the exit status object

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_duplicate_object	an exit status object with the same name already exists in the specified container
others	object id translation errors

os$create_job

```
(
OUT job_id : e$object_id;
IN object_parameters : e$object_parameters = DEFAULT;
IN job_record : e$job_record = DEFAULT;
IN job_initial_container : e$object_id = DEFAULT;
IN job_allocation_list : POINTER e$object_id_list = NIL;
IN process_object_parameters : e$object_parameters = DEFAULT;
IN process_record : e$process_record;
IN process_public_container : e$object_id = DEFAULT;
IN process_private_container : e$object_id = DEFAULT;
IN process_allocation_list : POINTER e$object_id_list = NIL;
IN process_data_block : POINTER quadword_data(*) CONFORM = NIL;
IN thread_object_parameters : e$object_parameters = DEFAULT;
IN thread_record : e$thread_record = DEFAULT;
IN thread_allocation_list : POINTER e$object_id_list = NIL;
IN thread_data_block : POINTER quadword_data(*) = NIL;
IN thread_immediate_parameter1 : POINTER anytype CONFORM = NIL;
IN thread_immediate_parameter2 : POINTER anytype CONFORM = NIL;
IN thread_status : e$object_id = DEFAULT;
) RETURNS status;
```

DESCRIPTION

Create a job, process, and thread object as specified by the parameters.

ARGUMENTS

job_id
Returns the object ID of the resulting job object

object_parameters
Supplies the object type independent parameters for the job object the ACL and container ID are ignored

job_record
Supplies the attributes of the job being created. If not present, then values are obtained from current user object

job_initial_container
Supplies the job level object container to be transfered into the job level container directory for this job. If not present then container directory comes up empty

job_allocation_list
Supplies the objects to be allocated to the job object. If not present then no objects are allocated to the job

process_object_parameters
Supplies the object type independent parameters for the process object the ACL and container ID are ignored

process_record
Supplies the attributes of the process being created

process_public_container
Supplies the process level public container to be transfered into the process level container directory for the process. If not present then the container comes up empty.

process_private_container
Supplies the process level private container to be transfered into the process level container directory for the process. If not present then container comes up empty.

process_allocation_list
Supplies the objects to be allocated to the process object. If not present then no objects are allocated to the process

process_data_block
Supplies an arbitrary data block passed to the process

thread_object_parameters
Supplies the object type independent parameters for the thread object the ACL and Container ID are ignored

thread_record
Supplies the attributes of the thread being created

thread_allocation_list
Supplies the objects to be allocated to the thread object. If not present then no objects are allocated to the thread

thread_data_block
Supplies an arbitrary data block passed to initial thread. Pointer in TCR, if pointer is NIL, then no data block was passed

thread_immediate_parameter1
Supplies an immediate parameter passed to thread through TCR

thread_immediate_parameter2
Supplies an immediate parameter passed to thread through TCR

thread_status
Supplies an exit status object to be bound to the initial thread. If not present then the thread is created without an exit status object

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_job_name_exists	a job object already exists with the name specified in the job object parameters
status$_bad_job_record	an invalid job record was specified
status$_bad_job_init_ container	the specfied job initial container can not be transfered to the new job
status$_bad_job_allocation	an invalid job allocation list was specified
status$_process_name_ exists	a process object already exists with the name specified in the process object parameters
status$_bad_process_record	an invalid process record was specified
status$_bad_prc_pub_ container	the specified process public container can not be transfered to the new process
status$_bad_prc_priv_ container	the specified process private container can not be transfered to the new process
status$_bad_process_ allocation	an invalid process allocation list was specified
status$_thread_name_exists	a thread object already exists with the name specified in the thread object parameters
status$_bad_thread_record	an invalid thread record was specified
status$_bad_thread_ allocation	an invalid thread allocation list was specified
status$_bad_process_exit_ status	an error occured translating the object id of the specified process exit status object
status$_bad_thread_exit_ status	an error occured translating the object id of the specified thread exit status object
status$_quota_exceeded	not enough quota exists to complete the service

os$create_process

```
(
OUT process_id : e$object_id;
IN object_parameters : e$object_parameters = DEFAULT;
IN process_record : e$process_record;
IN process_public_container : e$object_id = DEFAULT;
IN process_private_container : e$object_id = DEFAULT;
IN process_allocation_list : POINTER e$object_id_list = NIL;
IN process_data_block : POINTER quadword_data(*) CONFORM = NIL;
IN thread_object_parameters : e$object_parameters = DEFAULT;
IN thread_record : e$thread_record = DEFAULT;
IN thread_allocation_list : POINTER e$object_id_list = NIL;
IN thread_data_block : POINTER quadword_data(*) CONFORM = NIL;
IN thread_immediate_parameter1 : POINTER anytype CONFORM = NIL;
IN thread_immediate_parameter2 : POINTER anytype CONFORM = NIL;
IN thread_status : e$object_id = DEFAULT;
) RETURNS STATUS;
```

DESCRIPTION

Create a Process and thread object as specified by the parameters. Always results in the creation of a sub-process

ARGUMENTS

process_id
Returns the object ID of the resulting process object

object_parameters
Supplies the object type independent parameters for the process object the ACL and container ID are ignored

process_record
Supplies the attributes of the process being created

process_public_container
Supplies the process level public container to be transfered into the process level container directory for the process. If not present then the container comes up empty.

process_private_container
Supplies the process level private container to be transfered into the process level container directory for the process. If not present then container comes up empty.

process_allocation_list
Supplies the objects to be allocated to the process object. If not present then no objects are allocated to the process

process_data_block
Supplies an arbitrary data block passed to the process

thread_object_parameters
Supplies the object type independent parameters for the thread object the ACL and Container ID are ignored

thread_record
Supplies the attributes of the thread being created

thread_allocation_list
Supplies the objects to be allocated to the thread object. If not present then no objects are allocated to the thread

thread_data_block
Supplies an arbitrary data block passed to initial thread. Pointer in TCR, if pointer is NIL, then no data block was passed

thread_immediate_parameter1
Supplies an immediate parameter passed to thread through TCR

thread_immediate_parameter2
Supplies an immediate parameter passed to thread through TCR

thread_status
Supplies an exit status object to be bound to the initial thread. If not present then the thread is created without an exit status object

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_process_name_ exists	a process object already exists with the name specified in the process object parameters
status$_bad_process_record	an invalid process record was specified
status$_bad_prc_pub_ container	the specified process public container can not be transfered to the new process
status$_bad_prc_priv_ container	the specified process private container can not be transfered to the new process
status$_bad_process_ allocation	an invalid process allocation list was specified
status$_thread_name_exists	a thread object already exists with the name specified in the thread object parameters
status$_bad_thread_record	an invalid thread record was specified
status$_bad_thread_ allocation	an invalid thread allocation list was specified
status$_bad_process_exit_ status	an error occured translating the object id of the specified process exit status object
status$_bad_thread_exit_ status	an error occured translating the object id of the specified thread exit status object
status$_quota_exceeded	not enough quota exists to complete the service

os$create_thread

(
OUT thread_id : e$object_id;
IN object_parameters : e$object_parameters = DEFAULT;
IN thread_procedure : e$thread_entry_point;
IN thread_record : e$thread_record = DEFAULT;
IN thread_allocation_list : POINTER e$object_id_list = NIL;
IN thread_data_block : POINTER quadword_data(*) CONFORM = NIL;
IN thread_immediate_parameter1 : POINTER anytype CONFORM = NIL;
IN thread_immediate_parameter2 : POINTER anytype CONFORM = NIL;
IN thread_status : e$object_id = DEFAULT;
) RETURNS STATUS;

DESCRIPTION

Create and additional thread object as specified by the parameters.

ARGUMENTS

thread_id
Returns the object ID of the resulting process object

object_parameters
Supplies the object type independent parameters for the thread object the ACL and container ID are ignored

thread_procedure
Supplies the entrypoint for the new thread

thread_record
Supplies the attributes of the thread being created

thread_allocation_list
Supplies the objects to be allocated to the thread object. If not present then no objects are allocated to the thread

thread_data_block
Supplies an arbitrary data block passed to initial thread. Pointer in TCR, if pointer is NIL, then no data block was passed

thread_immediate_parameter1
Supplies an immediate parameter passed to thread through TCR

thread_immediate_parameter2
Supplies an immediate parameter passed to thread through TCR

thread_status
Supplies an exit status object to be bound to the initial thread. If not present then the thread is created without an exit status object

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_thread_name_exists	a thread object already exists with the name specified in the thread object parameters
status$_bad_thread_record	an invalid thread record was specified
status$_bad_thread_allocation	an invalid thread allocation list was specified
status$_bad_thread_exit_status	an error occured translating the object id of the specified thread exit status object
status$_quota_exceeded	not enough quota exists to complete the service

os$create_user

```
(
OUT user_id : e$object_id;
IN object_parameters : e$object_parameters = DEFAULT;
IN user_record : e$user_record;
IN user_allocation_list : POINTER e$object_id_list = NIL;
IN job_object_parameters : e$object_parameters = DEFAULT;
IN job_record : e$job_record = DEFAULT;
IN job_initial_container : e$object_id = DEFAULT;
IN job_allocation_list : POINTER e$object_id_list = NIL;
IN process_object_parameters : e$object_parameters = DEFAULT;
IN process_record : e$process_record;
IN process_public_container : e$object_id = DEFAULT;
IN process_private_container : e$object_id = DEFAULT;
IN process_allocation_list : POINTER e$object_id_list = NIL;
IN process_data_block : POINTER quadword_data(*) CONFORM = NIL;
IN thread_object_parameters : e$object_parameters = DEFAULT;
IN thread_record : e$thread_record = DEFAULT;
IN thread_allocation_list : POINTER e$object_id_list = NIL;
IN thread_data_block : POINTER quadword_data(*) CONFORM = NIL;
IN thread_immediate_parameter1 : POINTER anytype CONFORM = NIL;
IN thread_immediate_parameter2 : POINTER anytype CONFORM = NIL;
IN thread_status : e$object_id = DEFAULT;
) RETURNS STATUS;
```

DESCRIPTION

Create a user, job, process, and thread object as specified by the parameters. If the user object collides with an existing user object, then use the existing user object.

ARGUMENTS

user_id
Returns the object ID of the resulting user object

object_parameters
Supplies the object type independent parameters for the user object the ACL and container ID are ignored

user_record
Supplies the attributes of new user object.

user_allocation_list
Supplies the objects to be allocated to the user object. If not present then no objects are allocated to the user

job_object_parameters
Supplies the object type independent parameters for the job object the ACL and container ID are ignored

job_record
Supplies the attributes of the job being created. If not present, then values are obtained from current user object

job_initial_container
Supplies the job level object container to be transfered into the job level container directory for this job. If not present then container directory comes up empty

job_allocation_list
Supplies the objects to be allocated to the job object. If not present then no objects are allocated to the job

process_object_parameters
Supplies the object type independent parameters for the process object the ACL and container ID are ignored

process_record
Supplies the attributes of the process being created

process_public_container
Supplies the process level public container to be transfered into the process level container directory for the process. If not present then the container comes up empty.

process_private_container
Supplies the process level private container to be transfered into the process level container directory for the process. If not present then container comes up empty.

process_allocation_list
Supplies the objects to be allocated to the process object. If not present then no objects are allocated to the process

process_data_block
Supplies an arbitrary data block passed to the process

thread_object_parameters
Supplies the object type independent parameters for the thread object the ACL and Container ID are ignored

thread_record
Supplies the attributes of the thread being created

thread_allocation_list
Supplies the objects to be allocated to the thread object. If not present then no objects are allocated to the thread

thread_data_block
Supplies an arbitrary data block passed to initial thread. Pointer in TCR, if pointer is NIL, then no data block was passed

thread_immediate_parameter1
Supplies an immediate parameter passed to thread through TCR

thread_immediate_parameter2
Supplies an immediate parameter passed to thread through TCR

thread_status
Supplies an exit status object to be bound to the initial thread. If not present then the thread is created without an exit status object

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_bad_user_record	an invalid user record was specified
status$_bad_user_allocation	an invalid user allocation list was specified
status$_job_name_exists	a job object already exists with the name specified in the job object parameters
status$_bad_job_record	an invalid job record was specified
status$_bad_job_init_ container	the specfied job initial container can not be transfered to the new job
status$_bad_job_allocation	an invalid job allocation list was specified
status$_process_name_ exists	a process object already exists with the name specified in the process object parameters
status$_bad_process_record	an invalid process record was specified
status$_bad_prc_pub_ container	the specified process public container can not be transfered to the new process
status$_bad_prc_priv_ container	the specified process private container can not be transfered to the new process
status$_bad_process_ allocation	an invalid process allocation list was specified
status$_thread_name_exists	a thread object already exists with the name specified in the thread object parameters
status$_bad_thread_record	an invalid thread record was specified
status$_bad_thread_ allocation	an invalid thread allocation list was specified
status$_bad_process_exit_ status	an error occured translating the object id of the specified process exit status object
status$_bad_thread_exit_ status	an error occured translating the object id of the specified thread exit status object
status$_quota_exceeded	not enough quota exists to complete the service

os$delete_exit_handler_process

(
IN handler_id : e$exit_handler_id;
) RETURNS status;

DESCRIPTION

This service is used to delete an existing process level exit handler. The specified exit handler is removed from the process exit handler list. Once an exit handler is delete, it will not be processed.

ARGUMENTS

handler_id
Supplies the handler ID of the exit handler to be deleted.

RETURN VALUES

status$_normal	the service completed without errors
status$_exit_handler_not_found	the handler specified by handler_id was not found on exit handler list
status$_not_supported	an attempt to call this service from a system thread was made

os$delete_exit_handler_thread

(
IN handler_id : e$exit_handler_id;
) RETURNS status;

DESCRIPTION

This service is used to delete an existing thread level exit handler. The specified exit handler is removed from the threads exit handler list. Once an exit handler is deleted, it will not be processed.

ARGUMENTS

handler_id
Supplies the handler ID of the exit handler to be deleted.

RETURN VALUES

status$_normal	the service completed without errors
status$_exit_handler_not_found	the handler specified by handler_id was not found on exit handler list
status$_not_supported	an attempt to call this service from a system thread was made

os$exit_thread

(
IN exit_status : status;
) RETURNS status;

DESCRIPTION

This service begins kernel mode exit processing. This involves calling all thread level exit handlers. The thread object id is then removed. If the thread is the last thread in its process, then it executes its process level exit handlers.

ARGUMENTS

exit_status
Supplies the reason that the thread is exiting

RETURN VALUES

status$_repeat_service	Seen only by the system service dispatcher. This value is returned when dispatching to an exit handler. If the handler returns, os$exit_thread( ) is restarted.

os$force_exit_job

(
IN job_id : e$object_id;
IN exit_status : status;
) RETURNS status;

DESCRIPTION Force exit the job specified by job_id. This action causes all of the jobs processes to exit

ARGUMENTS

job_id
supplies object id of the job to be exited.

exit_status
supplies the reason for job to exit

RETURN VALUES

status$_normal	normal completion of the service
others	object id translation errors

os$force_exit_process

(
IN process_id : e$object_id;
IN exit_status : status;
) RETURNS status;

DESCRIPTION

Force exit the process specified by process_id. This action causes all of the processes sub-processes and threads to be force exited.

ARGUMENTS

process_id
Supplies the object id of the process to be exited.

exit_status
Supplies the reason for the process exiting

RETURN VALUES

status$_normal	normal completion of the service
others	object id translation errors

os$force_exit_thread

(
IN thread_id : e$object_id;
IN exit_status : status;
) RETURNS status;

DESCRIPTION Force exit the thread specified by thread_id.

ARGUMENTS

thread_id
supplies the object id of the thread to be exited.

exit_status
supplies the reason that the thread is force exiting

RETURN VALUES

status$_normal	normal completion of the service
others	object id translation errors

os$force_exit_user

(
IN user_id : e$object_id;
IN exit_status : status;
) RETURNS status;

DESCRIPTION

Force exit the user specified by user_obj_id. This action causes all of the users jobs to be force exited.

ARGUMENTS

user_id
Supplies the object id of the user to be exited.

exit_status
Supplies the reason for the user exiting

RETURN VALUES

status$_normal	normal completion of the service
others	object id translation errors

os$get_exit_status_info

(
IN exit_status_id : e$object_id = DEFAULT;
IN exit_status_items : POINTER e$item_list_type;
IN process_status_object : boolean = true;
) RETURNS status;

DESCRIPTION

Return information about the specified exit status. The information returned is item list driven

ARGUMENTS

exit_status_id
supplies the object id of the exit status object to get information from. If defaulted, then either the process exit status object of the current thread, or the thread exit status object of the current thread is assumed.

exit_status_items
supplies the item list which specifies the information to be retrieved.

Code	Pointer Type	Action
e$c_status_value	status	returns the status value from the item list
e$c_status_string	varying_string	returns the status string stored in the exit status object
e$c_status_string_set	boolean	returns and indication of whether a status string exists in the exit status object. True == exists
e$c_status_summary	e$exit_status_summary	returns the exit status summary from the exit status object. (this function does not return the status string, only its address has no use from user mode.)

process_status_object
only looked at if exits status id is defaulted. If true, the process level exit status object of the current thread is assumed, otherwise, the thread level exit status is assumed

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_invalid_item_code	a specified item code is invalid, or its item entry is invalid
others	object id translation errors

os$get_job_information

(
IN job_id : e$object_id = DEFAULT;
IN job_get_items : POINTER e$item_list_type;
) RETURNS status;

DESCRIPTION

Return information about the job object to the caller. The information returned is item list driven

ARGUMENTS

job_id
supplies if present, the object ID of job object that is to be inspected otherwise, the job object of the calling thread is assumed

job_get_items
supplies the item list identifying job object information to be extracted

Code	Pointer Type	Action
e$c_user_id	e$object_id	return the object id of the jobs user object
e$c_process_count	integer	return the number of processes for this user (subprocesss not included)
e$c_process_ids	e$object_id_list	return the object id's for the users processes (subprocesss not included)
e$c_quota_usage	e$quota_usage	return the jobs resource usage
e$c_job_limits	e$quota_limits	return the per job resource limits
e$c_job_class	e$job_class	return the job class of the job object

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_invalid_item_code	a specified item code is invalid, or its item entry is invalid
others	object id translation errors

os$get_process_information

(
IN process_id : e$object_id = DEFAULT;
IN process_get_items : POINTER e$item_list_type;
) RETURNS status;

DESCRIPTION

Return information about the process object to the caller. The information returned is item list driven

ARGUMENTS

process_id
supplies if present, the object ID of process object that is to be inspected otherwise, the process object of the calling thread is assumed

process_get_items
supplies the item list identifying process object information to be extracted

Code	Pointer Type	Action
e$c_job_id	e$object_id	return the object id of the processes job
e$c_parent_id	e$object_id	return the object id of the parent process zero( ) if process is not a subprocess
e$c_sub_process_count	integer	return the number of sub processes
e$c_sub_process_ids	e$object_id_list	return the object id's for the processes sub processes
e$c_thread_count	integer	return the number of threads for the process (threads in sub processes not included)
e$c_thread_ids	e$object_id_list	return the object ids for the threads of the process (threads in sub processes not included)
e$c_process_accounting	e$accounting_summary	return the process level accounting summary
e$c_pcr_base	e$process_control_region	return address of the process control region
e$c_quota_usage	e$quota_usage	return the processes resource usage
e$c_process_limits	e$quota_limits	return the per process resource limits

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_invalid_item_code	a specified item code is invalid, or its item entry is invalid
others	object id translation errors

os$get_thread_information

(
IN thread_id : e$object_id = DEFAULT;
IN thread_get_items : POINTER e$item_list_type;
) RETURNS status;

DESCRIPTION

Return information about the thread object to the caller. The information returned is item list driven

ARGUMENTS

thread_id
supplies if present, the object ID of thread object that is to be inspected otherwise, the thread object of the calling thread is assumed

thread_get_items
supplies the item list identifying thread object information to be extracted

Code	Pointer Type	Action
e$c_process_id	e$object_id	returns the object id of the threads process
e$c_tcr_base	e$thread_control_region	returns address of the threads tcr
e$c_thread_accounting	e$cpu_and_io_summary	returns the thread specific accounting summary
e$c_thread_perf_counters	e$thread_perf_counters	returns the thread performance counters
e$c_thread_priority	k$combined_priority	return the current thread priority
e$c_thread_affinity	k$affinity	return the current thread affinity

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_invalid_item_code	a specified item code is invalid, or its item entry is invalid
others	object id translation errors

os$get_user_information

(
IN user_id : e$object_id = DEFAULT;
IN user_get_items : POINTER e$item_list_type;
) RETURNS status;

DESCRIPTION

Return information about the user object to the caller. The information returned is item list driven

ARGUMENTS

user_id
supplies if present, the object ID of user object that is to be inspected otherwise, the user object of the calling thread is assumed

user_get_items
supplies the item list identifying user object information to be extracted

Code	Pointer Type	Action
e$c_job_count	integer	return the number of jobs for this user
e$c_job_ids	e$object_id_list	return the object id's for the users jobs
e$c_username	varying_string	return the user name
e$c_quota_usage	e$quota_usage	return the users resource usage
e$c_user_limits	e$quota_limits	return the users resource limits
e$c_job_limits	e$quota_limits	return the per job resource limits
e$c_process_limits	e$quota_limits	return the per process resource limits
e$c_thread_priority	k$combined_priority	return the default thread priority
e$c_thread_affinity	k$affinity	return the default thread affinity
e$c_access_restrictions	e$access_restrictions	return the access retrictions

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_invalid_item_code	a specified item code is invalid, or its item entry is invalid
others	object id translation errors

os$hibernate_process

(
IN process_id : e$object_id;
) RETURNS status;

DESCRIPTION

Cause all threads owned by the process specified by process_id to issue a wait on the auto-clearing hibernate event object in their TCB. User mode AST's remain enabled

ARGUMENTS

process_id
supplies the object of the target process

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_quota_exceeded	not enough quota exists to capture the thread or subprocess ids of the specified process
others	object id translation errors

os$hibernate_thread

(
IN thread_id : e$object_id;
) RETURNS status;

DESCRIPTION

Cause the thread specified by thread_id to issue a wait on the auto-clearing hibernate event object in its TCB. User mode AST's remain enabled

ARGUMENTS

thread_id
supplies the object of the target thread

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
others	object id translation errors

os$resume_process

(
IN process_id : e$object_id;
) RETURNS status;

DESCRIPTION

Cause all threads owned by the process specified by process object_id to have their waits on the auto-clearing suspend event object in their TCB to be satisfied by setting the event.

ARGUMENTS

process_id
supplies the object ID of the target process

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_quota_exceeded	not enough quota exists to capture the thread or subprocess ids of the specified process
others	object id translation errors

os$resume_thread

(
IN thread_id : e$object_id;
) RETURNS status;

DESCRIPTION

Cause the thread specified by thread object_id to have its wait on the auto-clearing suspend event object in its TCB to be satisfied by setting the event.

ARGUMENTS

thread_id
supplies the object ID of the target thread

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
others	object id translation errors

os$set_exit_status_info

(
IN exit_status_id : e$object_id = DEFAULT;
IN exit_status_items : POINTER e$item_list_type;
IN process_status_object : boolean = true;
) RETURNS status;

DESCRIPTION

Set information in the specified exit status. The information returned is item list driven

ARGUMENTS

exit_status_id
supplies the object id of the exit status object to set information into. If defaulted, then either the process exit status object of the current thread, or the thread exit status object of the current thread is assumed. When this id is defaulted, then the process or thread level exit status object is used by address (no acl protection) since we assume that you can always write to your own exit status object.

exit_status_items
supplies the item list which specifies the information to be set.

Code	Pointer Type	Action
e$c_status_string	varying_string	places the specified string in the exit status object

process_status_object
only looked at if exits status id is defaulted. If true, the process level exit status object of the current thread is assumed, otherwise, the thread level exit status is assumed

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_invalid_item_code	a specified item code is invalid, or its item entry is invalid
others	object id translation errors

os$set_job_information

(
IN job_id : e$object_id = DEFAULT;
IN job_set_items : POINTER e$item_list_type;
) RETURNS status;

DESCRIPTION

Return information about the job object to the caller. The information returned is item list driven

ARGUMENTS

job_id
supplies if present, the object ID of job object that is to be modified otherwise, the job object of the calling thread is assumed

job_set_items
supplies the item list identifying job object information to be modified

Code	Pointer Type	Action
e$c_job_limits	e$quota_limits	set the per job resource limits

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_invalid_item_code	a specified item code is invalid, or its item entry is invalid
others	object id translation errors

os$set_minor_thread_priority

(
IN thread_id : e$object_id = DEFAULT;
IN new_priority : k$minor_priority;
OUT previous_priority : k$combined_priority;
) RETURNS status;

DESCRIPTION

This system service changes the minor priority of the specified thread.

ARGUMENTS

thread_id
Supplies the object id of the thread whose priority is to be altered. If this parameter is defaulted, the current thread is assumed

new_priority
Supplies the minor priority that is to be set in the specified thread.

previous_priority
Returns the specified threads previous combined priority. Only valid if status$_normal was returned.

RETURN VALUES

status$_normal	the service completed without errors
status$_invalid_argument	new_priority is not a valid value for k$minor_priority
others	object id translation errors

os$set_process_information

(
IN process_id : e$object_id = DEFAULT;
IN process_set_items : POINTER e$item_list_type;
) RETURNS status;

DESCRIPTION

Return information about the process object to the caller. The information returned is item list driven

ARGUMENTS

process_id
supplies if present, the object ID of process object that is to be modified otherwise, the process object of the calling thread is assumed

process_set_items
supplies the item list identifying process object information to be modified

Code	Pointer Type	Action
e$c_protected_data	anytype	add block to protected data listhead in the pcr (item length determines how many bytes of data are being linked to the list.)
e$c_process_limits	e$quota_limits	replace the per process resource limits

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_invalid_item_code	a specified item code is invalid, or its item entry is invalid
others	object id translation errors

os$set_thread_information

(
IN thread_id : e$object_id = DEFAULT;
IN thread_set_items : POINTER e$item_list_type;
) RETURNS status;

DESCRIPTION

Return information about the thread object to the caller. The information returned is item list driven

ARGUMENTS

thread_id
supplies if present, the object ID of thread object that is to be modified otherwise, the thread object of the calling thread is assumed

thread_set_items
supplies the item list identifying thread object information to be modified

Code	Pointer Type	Action
e$c_thread_priority	k$combined_priority	set the current thread priority
e$c_thread_mnr_priority	k$minor_priority	set the current thread minor priority
e$c_thread_mjr_priority	k$major_priority	set the current thread major priority
e$c_thread_affinity	k$affinity	set the current thread affinity

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_invalid_item_code	a specified item code is invalid, or its item entry is invalid
others	object id translation errors

os$set_thread_priority

(
IN thread_id : e$object_id = DEFAULT;
IN new_priority : k$combined_priority = 0;
OUT previous_priority : k$combined_priority;
) RETURNS status;

DESCRIPTION

This system service changes the combined priority of the specified thread.

ARGUMENTS

thread_id

Supplies the object id of the thread whose priority is to be altered. If this parameter is defaulted, the current thread is assumed

new_priority

Supplies the combined priority that is to be set in the thread. If this parameter is defaulted, the base priority of the threads process is assumed. If the major priority in new_priority is greater than the threads current major priority, then the calling thread must have access to the raise priority privileged operation object.

This service never allows the priority to be changed out of the priority class that the thread process is a member of. If the process is not in a realtime priority class, then the threads priority can not be changed to a realtime priority class. If the process is within a realtime priority class, then the threads new priority must stay within a realtime priority class.

previous_priority

Returns the specified threads previous combined priority. Only valid if status$_normal was returned.

RETURN VALUES

status$_normal	the service completed without errors
status$_invalid_argument	new_priority is not a valid value for k$combined_ priority, or specifies a priority class that is different from the threads process
others	object id translation errors

os$set_user_information

(
IN user_id : e$object_id = DEFAULT;
IN user_set_items : POINTER e$item_list_type;
) RETURNS status;

DESCRIPTION

Return information about the user object to the caller. The information returned is item list driven

ARGUMENTS

user_id
supplies if present, the object ID of user object that is to be modified otherwise, the user object of the calling thread is assumed

user_set_items
supplies the item list identifying user object information to be modified

Code	Pointer Type	Action
e$c_user_limits	e$quota_limits	set the users resource limits
e$c_job_limits	e$quota_limits	set the per job resource limits
e$c_process_limits	e$quota_limits	set the per process resource limits
e$c_thread_priority	k$combined_priority	set the default thread priority
e$c_thread_affinity	k$affinity	set the default thread affinity
e$c_access_restrictions	e$access_restrictions	set the access retrictions

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_invalid_item_code	a specified item code is invalid, or its item entry is invalid
others	object id translation errors

os$signal_process

(
IN process_id : e$object_id;
IN condition_value : status;
IN signal_argument : longword CONFORM = DEFAULT;
) RETURNS status;

DESCRIPTION

Cause a condition of type condition_value to be raised in all threads owned by the process specified by process_id. The condition handler is passed signal_argument.

ARGUMENTS

process_id
supplies the object_id of the process to be signaled

condition_value
supplies a condition value to be raised in all threads of the target process

signal_argument
supplies the value that is passed to the condition handler

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
others	object id translation errors

os$signal_thread

(
IN thread_id : e$object_id;
IN condition_value : status;
IN signal_argument : longword CONFORM = DEFAULT;
) RETURNS status;

DESCRIPTION

Cause a condition of type condition_value to be raised in the thread specified by thread_id. The condition handler is passed signal_argument.

ARGUMENTS

thread_id
supplies the object_id of the thread to be signaled

condition_value
supplies a condition value to be raised in all threads of the target thread

signal_argument
supplies the value that is passed to the condition handler

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_not_supported	the target thread was a system thread
others	object id translation errors

os$suspend_process

(
IN process_id : e$object_id;
) RETURNS status;

DESCRIPTION

Cause all threads owned by the process specified by process_id to issue a wait on the auto-clearing suspend event object in their TCB. User mode AST's are disabled.

ARGUMENTS

process_id
supplies the object ID of the target process

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_quota_exceeded	not enough quota exists to capture the thread or subprocess ids of the specified process
others	object id translation errors

os$suspend_thread

(
IN thread_id : e$object_id;
) RETURNS status;

DESCRIPTION

Cause the thread specified by thread_id to issue a wait on the auto-clearing suspend event object in its TCB. User mode AST's are disabled.

ARGUMENTS

thread_id
supplies the object ID of the target thread

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
others	object id translation errors

os$wake_process

*(
IN process_id : e$object_id;
) RETURNS status;*

DESCRIPTION

Cause all threads owned by the process specified by process_id to have their waits on the auto-clearing hibernate event object in their TCB to be satisfied by setting the event.

ARGUMENTS

process_id
supplies the object ID of the target process

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
status$_quota_exceeded	not enough quota exists to capture the thread or subprocess ids of the specified process
others	object id translation errors

os$wake_thread

(
IN thread_id : e$object_id;
) RETURNS status;

DESCRIPTION Cause the thread specified by thread_id to have its wait on the auto-clearing hibernate event object in its TCB to be satisfied by setting the event.

ARGUMENTS

thread_id
supplies the object ID of the target thread

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessable
others	object id translation errors

8 Memory System Services

os$adjust_working_set_limit

(
IN number_of_bytes : integer;
OUT new_working_set_limit : integer [1..];
) RETURNS STATUS;

DESCRIPTION

The Adjust Working Set Limit service adjusts a process's current working set limit by the specified number of bytes and returns the new value to the caller. The specified number of bytes will be converted into pages and the calculated number of pages will be added to or removed from the working set. A negative value for the byte count will cause pages to be removed from the working set.

ARGUMENTS

number_of_bytes
Supplies the number of bytes to add or remove from the working set.

new_working_set_limit
Returns the current size of the working set in bytes. The working set is maintained in pages and converted to bytes.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_address	error, either the starting or ending address is not accessable.
status$_working_set_at_ maximum	error, unable to add any more pages to the working set.
status$_working_set_at_ minimum	error, unable to remove any more pages from the working set.

os$create_address_space

(
IN desired_beginning_address : POINTER anytype CONFORM;
IN desired_ending_address : POINTER anytype CONFORM;
OUT actual_beginning_address : POINTER anytype CONFORM;
OUT actual_ending_address : POINTER anytype CONFORM;
) RETURNS status;

DESCRIPTION

This routine creates address space at the specified address. An error is returned if any of the desired address range is already mapped, but the create address will map from the desired address up to the already created addresses, and that range will be returned.

ARGUMENTS

desired_beginning_address
Supplies the beginning address of the range to create.

desired_ending_address
Supplies the ending address of the range to create.

actual_beginning_address
Returned address of the beginning of the range actually created. The actual range could differ from the desired range due to 64K byte alignment.

actual_ending_address
Returned address of the ending of the range actually created.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_begin_ address	error, the beginning address is invalid.
status$_invalid_ending_ address	error, the ending address is invalid.
status$_complete_range_ not_map	warning, the complete range of addresses could not be mapped do to previously mapped addresses.

os$create_section

(
OUT section_id : e$object_id;
IN object_parameters : e$object_parameters = DEFAULT;
IN file_channel : integer OPTIONAL; !### needs fixed also item list needs to be added
IN mapping_type : e$mapping_type OPTIONAL;
IN size_in_bytes : integer OPTIONAL;
IN virtual_block_number : integer OPTIONAL;
IN protection : e$page_protection OPTIONAL;
IN identification_match : integer OPTIONAL;
) RETURNS status;

DESCRIPTION

This routine creates a section which is either backed by an existing file or backed by paging file.

ARGUMENTS

section_id
Returned object ID of the created section.

object_parameters
Supplies the object container in which the object is inserted, the name of the object, and the access control list (ACL) of the object. If this argument is not supplied or if it is supplied but not all values in the object parameter record are supplied, the service applies default values. The default object container is the process private container, the default name is none, and the default ACL is none. to map the section into.

file_channel
Supplies the object ID of a previously created channel which has had a file open performed. If the channel is not supplied, a section backed by paging file is created.

mapping_type
Supplies the type of section to create, either data or image.

size_in_bytes
Supplies the size of the section to create in bytes. If page file mapping is performed this parameter is required.

virtual_block_number
Supplies the virtual block number offset within the opened file to begin mapping. This virtual block number is aligned on a 64K byte boundary. Hence is the virtual block number is specified as 40 the actual virtual block number would be 33 (start at vbn 1).

protection
Supplies the desired protection to apply to the newly created pages, optional.

identification_match
Supplies the id to match, optional.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_address	error, either the starting or ending address is not accessable.
status$_mapping_conflict	error, the specified address range contains pages which are already mapped.
status$_invalid_section_size	error, the size specified for the section is invalid.
status$_requires_channel_ arg	error, the section type requires a channel to be specified.
others	any object error in creating an object.

os$delete_address_space

(
IN desired_beginning_address : POINTER anytype CONFORM;
IN desired_ending_address : POINTER anytype CONFORM;
OUT actual_beginning_address : POINTER anytype CONFORM;
OUT actual_ending_address : POINTER anytype CONFORM;
) RETURNS status;

DESCRIPTION

This routine deletes the address space at the specified address. An warning status is returned if any of the desired address range is mapped in by a mapping object, i.e. was not created by e$create_virtual_address_space and only the address space up to the found address is deleted.

ARGUMENTS

desired_beginning_address
Supplies the beginning address of the range to delete.

desired_ending_address
Supplies the ending address of the range to delete.

actual_beginning_address
Returned address of the beggin of the range actually deleted. The actual range could differ from the desired range due to 64K byte alignment.

actual_ending_address
Returned address of the ending of the range actually deleted.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_begin_address	error, the beginning address is invalid.
status$_invalid_ending_address	error, the ending address is invalid.
status$_total_range_not_deleted	warning, the complete range of addresses could not be deleted do to previously mapped addresses.

os$expand_address_space

(
IN number_of_bytes : integer [0..];
OUT actual_beginning_address : POINTER anytype CONFORM;
OUT actual_ending_address : POINTER anytype CONFORM;
) RETURNS status;

DESCRIPTION

This routine creates address space starting at the highest virtual address in use by the process for the number of bytes specified.

ARGUMENTS

number_of_bytes
Supplies the number of bytes to add to the address space.

actual_beginning_address
Returned address of the first byte of the created address range.

actual_ending_address
Returned address of the last byte of the created address range.

RETURN VALUES

status$_normal	normal, successful completion.
status$_complete_range_ not_map	warning, the complete range of addresses could not be mapped do to previously mapped addresses.

os$expand_user_stack

(
IN number_of_bytes_to_add : integer [1..];
OUT new_stack_size : integer [1..];
) RETURNS STATUS;

DESCRIPTION

The Expand User Stack service attempts to adjust the user stack by the specified number of bytes. The number of bytes is converted into pages and an attempt is made to expand the stack by the calculated number of pages.

The stack expansion may fail due to other thead user stacks occupying virtual address space and thereby preventing the stack expansion. Note that there is no way to contract a stack.

ARGUMENTS

number_of_bytes_to_add
Supplies the number of bytes to add to the stack. The number of bytes is converted to pages.

new_stack_size
Returns the current stack size in bytes.

RETURN VALUES

status$_normal	normal, successful completion.
status$_unable_to_expand_stack	error, stack expansion failed.
status$_partial_expansion	warning, not all bytes were added to the stack.
status$_invalid_address	error, either the starting or ending address is not accessable.

os$get_mapping_information

(
IN mapping_id : e$object_id;
IN mapping_get_items : POINTER e$item_list_type;
) RETURNS STATUS;

DESCRIPTION

The Get Mapping Information service provides information about the specified mapping object. The information which may be obtained is specified in an item list.

ARGUMENTS

mapping_id

Supplies the object ID of the desired mapping object on which information should be extracted.

mapping_get_items

Supplies the item list which specifies the information about the mapping object to return.

item code	description
e$c_mapping_section	The object ID of the section which this mapping object maps.
e$c_mapping_starting_ address	The starting address of the mapping in the address space.
e$c_mapping_size	The size of the mapping in bytes.
e$c_mapping_offset	The byte offset from the start of the section object.

RETURN VALUES

status$_normal	normal, successful completion.
object_reference_errors	any errors trying to reference an object by id.

os$get_section_information

(
IN section_id : e$object_id;
IN section_get_items : POINTER e$item_list_type;
) RETURNS STATUS;

DESCRIPTION

The Get Section Information service provides information about the specified section object. The information which may be obtained is specified in an item list.

ARGUMENTS

section_id
Supplies the object ID of the desired section on which information should be extracted.

section_get_items
Supplies the item list which specifies the information about the section to return.

The following codes are valid:

item code	action
e$c_section_vbn	Virtual block number offset which the section is based upon.
e$c_section_size	Size of the section in bytes.
e$c_section_protection_code	Protection code assigned to section pages.
e$c_section_ident_match	Identification match specified on section.
e$c_section_type	Type of section (image or data).

RETURN VALUES

status$_normal	normal, successful completion.
object_reference_errors	any errors trying to reference an object by id.

os$lock_pages_in_memory

(
IN starting_address : POINTER anytype CONFORM;
IN ending_address : POINTER anytype CONFORM;
OUT last_locked_address : POINTER anytype CONFORM;
) RETURNS STATUS;

DESCRIPTION

The Lock Pages in Memory service locks a page or range of pages in memory. The specified virtual pages are forced into the working set, then locked in memory. A locked page is not removed from memory if its process's working set is removed from the balance set.

ARGUMENTS

starting_address
Supplies the starting virtual address of the range to be locked into memory.

ending_address
Supplies the ending virtual address of the the range to be locked into memory.

last_locked_address
Returns the last address which was actually locked in memory.

RETURN VALUES

status$_normal	normal, successful completion.
status$_complete_range_not_lock	warning, at least one page was locked in memory.
status$_locked_limit_reached	error, no more pages may be locked in memory.
status$_invalid_address	error, either the starting or ending address is not accessable.

os$lock_pages_working_set

(
IN starting_address : POINTER anytype CONFORM;
IN ending_address : POINTER anytype CONFORM;
OUT last_locked_address : POINTER anytype CONFORM;
) RETURNS STATUS;

DESCRIPTION

The lock pages in working set service locks a page or range of pages in a process's working set. The specified virtual pages are forced into the working set.

ARGUMENTS

starting_address
Supplies the starting virtual address of the range to be locked into the working set.

ending_address
Supplies the ending virtual address of the the range to be locked into the working set.

last_locked_address
Returns the last address which was actually locked in the working set.

RETURN VALUES

status$_normal	normal, successful completion.
status$_complete_range_ not_lock	warning, at least one page was locked in the working set.
status$_working_set_full	error, no more pages may be locked in the working set.
status$_invalid_address	error, either the starting or ending address is not accessable.

os$map_section

(
OUT mapping_id : e$object_id;
IN object_parameters : e$object_parameters = DEFAULT;
IN section_id : e$object_id;
IN desired_beginning_address : POINTER anytype CONFORM OPTIONAL;
IN desired_ending_address : POINTER anytype CONFORM OPTIONAL;
IN protection : e$page_protection OPTIONAL;
IN identification_match : integer OPTIONAL;
IN byte_offset : integer [0..] OPTIONAL;
OUT actual_beginning_address : POINTER anytype CONFORM;
OUT actual_ending_address : POINTER anytype CONFORM;
) RETURNS status;

DESCRIPTION

This routine maps a previously created section into the process's address space.

ARGUMENTS

mapping_id
Returned object ID of the mapping object which describes the memory section.

object_parameters
Supplies the object container in which the object is inserted, the name of the object, and the access control list (ACL) of the object. If this argument is not supplied or if it is supplied but not all values in the object parameter record are supplied, the service applies default values. The default object container is the process private container, the default name is none, and the default ACL is none.

section_id
Supplies the object ID of previously created section.

desired_beginning_address
Supplies the beginning address of the range to map the section into. The range must not currently have any valid addresses. The actual mapping occurs on a 64K bytes boundary.

desired_ending_address
Supplies the ending address of the range to map the section into.

protection
Supplies the desired protection to apply to the newly created pages, optional.

identification_match
Supplies the id to match, optional.

byte_offset
Supplies the offset into the section to beginning mapping, optional.

actual_beginning_address
Returns the actual beginning address of the created range.

actual_ending_address
Returns the actual ending address of the created range.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_address	error, either the starting or ending address is not accessable.
status$_mapping_conflict	error, the specified address range contains pages which are already mapped.
status$_invalid_map_ container	error, the specified container for the mapping object was not the default private container.
others	any object error in creating an object.

os$set_protection_on_pages

(
IN starting_address : POINTER anytype CONFORM;
IN ending_address : POINTER anytype CONFORM;
IN page_protection : e$page_protection;
OUT last_changed_address : POINTER anytype CONFORM;
OUT previous_page_protection : e$page_protection OPTIONAL;
) RETURNS status;

DESCRIPTION

The Set Protection on Pages system service allows a thread to change the protection on a page or range of pages.

ARGUMENTS

starting_address
Supplies the starting virtual address of the range to have its protection modified.

ending_address
Supplies the ending virtual address of the the range to have its protection modified.

page_protection
Supplies the page protection to assign to the pages within the specified address range. The page protection is a set with the following members. Note that write implies read and for user access, kernel access is always set to be identical. Also, user execute or kernel execute implies the other.

protection code	protection
e$c_page_user_read	user read access.
e$c_page_user_write	user write,read access.
e$c_page_user_execute	user execute access.
e$c_page_kernel_read	kernel read access.
e$c_page_kernel_write	kernel write access.
e$c_page_kernel_execute	kernel execute access.

last_changed_address
Returns the last address which the protection was actually changed.

previous_page_protection
Optionally returns the previous page protection for the first page which the protection was actually changed.

RETURN VALUES

status$_normal	normal, sucessful completion.
status$_partial_range_done	warning, unable to change the protection on the complete range do to nonexistant pages.
status$_invalid_argument	error, unable to access or iterpret argument.
status$_invalid_protection	error, protection set contains invalid members.
status$_page_owner_ violation	error, attempt to change kernel protection on kernel owned pages.

os$unlock_pages_from_memory

(
IN starting_address : POINTER anytype CONFORM;
IN ending_address : POINTER anytype CONFORM;
OUT last_unlocked_address : POINTER anytype CONFORM;
) RETURNS STATUS;

DESCRIPTION

The unlock pages from memory service unlocks a page or range of pages from memory. The specified virtual pages are unlocked from memory and become eligible for replacement.

ARGUMENTS

starting_address
Supplies the starting virtual address of the range to be unlocked from memory.

ending_address
Supplies the ending virtual address of the the range to be unlocked from memory.

last_locked_address
Returns the last address which was actually unlocked from memory.

RETURN VALUES

status$_normal	normal, successful completion.
status$_complete_range_ not_lock	warning, at least one page was unlocked from memory.
status$_invalid_address	error, either the starting or ending address is not accessable.

os$unlock_pages_working_set

(
IN starting_address : POINTER anytype CONFORM;
IN ending_address : POINTER anytype CONFORM;
OUT last_unlocked_address : POINTER anytype CONFORM;
) RETURNS STATUS;

DESCRIPTION

The unlock pages from working set service unlocks a page or range of pages from a process's working set. The specified virtual pages are unlocked from the working set and become eligible for replacement.

ARGUMENTS

starting_address
Supplies the starting virtual address of the range to be unlocked from the working set.

ending_address
Supplies the ending virtual address of the the range to be unlocked from the working set.

last_locked_address
Returns the last address which was actually unlocked from the working set.

RETURN VALUES

status$_normal	normal, successful completion.
status$_complete_range_ not_lock	warning, at least one page was unlocked in the working set.
status$_invalid_address	error, either the starting or ending address is not accessable.

os$update_mapped_section

(
IN mapping_id : e$object_id;
IN desired_beginning_address : POINTER anytype CONFORM;
IN desired_ending_address : POINTER anytype CONFORM;
IN flags : e$section_update_flags;
IN event_id : e$object_id OPTIONAL;
IN ast_procedure : k$normal_ast_routine OPTIONAL;
IN ast_parameter : LONGWORD CONFORM OPTIONAL;
BIND io_status_block : e$iosb;
OUT actual_beginning_address : POINTER anytype CONFORM;
OUT actual_ending_address : POINTER anytype CONFORM;
) RETURNS STATUS;

DESCRIPTION

The Update Mapped Section service writes all modified pages in a mapped section back into the section file on disk. One or more I/O requests are queued based on the number of pages that have been modified.

ARGUMENTS

mapping_id
Supplies the mapping ID of the mapped section to update.

desired_beginning_address
Optionally supplies the beginning address within the mapping to begin updating the section. If this argument is not specified, the starting address of the mapping will be used.

desired_ending_address
Optionally supplies the ending address within the mapping to end updating the section. If this argument is not specified, the endinng address of the mapping will be used.

flags
Optionally supplies the update specified for updating the section. More here later.

event_id
Optionally supplies the object ID of an event object which will be set when the update operation has completed.

ast_procedure
Optionally supplies the address of an AST procedure which will be called when the update operation has completed.

ast_parameter
Optionally supples the value which will be supplied to the AST procedure when called.

io_status_block
Optionally supplies the I/O status block which will receive the final completion status of the updating operation.

actual_beginning_address
Optionally returns the actual beginning address of the update operation.

actual_ending_address
Optionally returns the actual ending address of the update operation.

RETURN VALUES

status$_normal	normal, sucessful completion.
status$_invalid_address_range	error, beginning or ending address was not within the mapping as specified by the mapping ID.
object_reference_errors	any errors trying to reference an object by id.

os$zero_to_end_of_user_stack

(
) RETURNS STATUS;

DESCRIPTION	The Zero to End of User Stack service zeroes all pages from the current stack pointer to the end of the stack. The zeroing is accomplished by releasing any pages in physical memory or in the paging file and converting the pages into demand zero pages.
ARGUMENTS	***None.***

RETURN VALUES	status$_normal	normal, successful completion.

9 I/O System Services

os$cancel_io

(
IN channel_id : e$object_id;
) RETURNS status;

DESCRIPTION

This service cancels all outstanding I/O request on the specified channel. Only the outstanding I/O requests that were issued by the calling thread are canceled.

Outstanding I/O requests that are canceled are done so, asynchronously to the the completion of the this service. That is, completion of this service cannot be used to synchronize with the cancellation of the I/O requests.

ARGUMENTS

channel_id
Supplies an ID of the channel

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id
status$_object_type_ mismatch	invalid object

os$configure_fp

(
IN fpd_id : e$object_id;
IN function_code : integer;
IN user_event : e$object_id = DEFAULT;
IN fpd_parameters : POINTER anytype CONFORM = DEFAULT;
) RETURNS status;

DESCRIPTION

This service is used to issue configuration and deconfiguration requests to a function processor. The function code and the fpd_parameters specifies the reqeust type.

The user supplied event object is specified if the caller wants to synchronized with the completion of the request.

ARGUMENTS

fpd_id
Supplies the FPD object ID

function_code
Supplies the configuration function code

user_event
Supplies object id of event to be signalled when done

fpd_parameters
Supplies the FPD configuration parameters.

RETURN VALUES

status$_normal	normal, successful completion
status$_invalid_object_id	invalid object id
status$_object_type_mismatch	invalid object

os$create_channel

*(
OUT channel_id : e$object_id;
IN object_parameters : e$object_parameters;
IN fpu_id : e$object_id;
) RETURNS status;*

DESCRIPTION

This service is call to create a channel to an existing FPU object. The FPU object ID parameter specifies the FPU object to which the channel is attach.

The object ID of the newly created channel is returned in the channel_ id parameter. After the channel object is created it is inserted into the container specified in the object_parameters record. If there is a duplicate object currently in the container, the newly created channel object is deleted, and the object ID of the duplicate object is returned. If a container object ID is not specified, the channel object is placed in the process private container.

ARGUMENTS

channel_id
Returns a channel id

object_parameters
Supplies the object architecture create object parameters

fpu_id
Supplies an object id of the FPU object to create a channel to

RETURN VALUES

status$_normal	normal, successful completion
status$_invalid_object_id	invalid object id
status$_duplicate_object	duplicate object found in object container
status$_object_contianer_full	object container full
status$_object_type_ mismatch	invalid object

os$create_fpu

(
OUT fpu_id : e$object_id;
IN object_parameters : e$object_parameters;
IN fpd_id : e$object_id;
IN fpu_parameters : POINTER anytype CONFORM = DEFAULT;
) RETURNS status;

DESCRIPTION

This service creates an FPU object for a function processor. The fpd_id parameter specifies the function processor for which the FPU object is created for.

The object ID of the newly created FPU object is returned in the fpu_id parameter. The object parameters specifies the object name, an ACL for the FPU object, and the object ID of the container where the FPU object is to be inserted in.

If a container object ID is not supplied, the FPU object is inserted into the process private container after it is created. If a duplicate object already exist in the specified container, the newly created FPU object is deleted, and the object ID of the duplicate object is returned

ARGUMENTS

fpu_id
Return the object id of the created FPU object.

object_paramters
Supplies the object parameters.

fpd_id
Supplies the object id of fpd.

fpu_parameters
Supplies the FPU specific parameters used to initialize the the FPU object.

RETURN VALUES

status$_normal	normal, successful completion
status$_invalid_object_id	invalid object
status$_duplicate_object	duplicate object found in object container
status$_object_container_full	object container full

os$get_channel_information

(
IN channel_id : e$object_id;
IN channel_items : POINTER e$item_list_type = DEFAULT;
) RETURNS status;

DESCRIPTION Returns information about a channel object. The information returned is item list driven.

ARGUMENTS

channel_id
Supplies channel object ID.

channel_items
Supplies a pointer to an item list.

Item Codes	Data Type	Description
io$c_item_channel_access	BOOLEAN	TRUE, if channel is being access.
io$c_item_granted_access	SET[access_type]	Returns the access types that have been granted on this channel.

RETURN VALUES

status$_normal	normal, successful completion
status$_invalid_object_id	invalid object id
status$_object_type_ mismatch	invalid object

os$get_fpu_information

*(
IN fpu_id : e$object_id;
IN fpu_items : POINTER e$item_list_type = DEFAULT;
) RETURNS status;*

DESCRIPTION

Returns information about an FPU object. The information returned is item list driven.

ARGUMENTS

fpu_id
Supplies an FPU object ID.

fpu_items
Supplies a pointer to an item list.

Item Codes	Data Type	Description
io$c_item_interface_class	INTEGER	Returns FPU interface class
io$c_item_fpu_state	e$fpu_state	FPU current state
io$c_fpu_bound	Integer	Returns TRUE if FPU is bound
io$c_item_fp_params_area_size	Integer	Returns size of the FP parameter area needed by this function processor and all function processor below it. The size is returned in quadwords.

RETURN VALUES

status$_normal	normal, successful completion
status$_invalid_object_id	invalid object ID
status$_object_type_mismatch	invalid object

os$request_io

(
IN channel_id : e$object_id;
IN function_code : integer;
BIND iosb : e$iosb;
IN completion_event_id : e$object_id = DEFAULT;
IN completion_ast : k$normal_ast_routine = DEFAULT;
IN ast_parameter : POINTER anytype CONFORM = DEFAULT;
IN io_parameters : POINTER anytype CONFORM = DEFAULT;
) RETURNS status;

DESCRIPTION

This service is used to issue an I/O request. Two types of I/O request may be issued, they are:

a. Asynchronous I/O request, and

b. Synchronous I/O request

An I/O request is describe by its function code and I/O parameter record supplied to this service. The request will fail if the channel or event object is invalid, the function code or I/O parameters are invalid. The returned status will contain the cause of failure. No information will be written to the I/O status block.

An asynchronous I/O request is issued if an event object, AST procedure, or both are specified in the call. Control is return to the caller after the request has been successfully posted. When the I/O completes, the following events can occur:

a. If an event object was specified, it is signalled.

b. If an AST procedure was specified, the AST is queued to the calling thread.

c. If both event object and a AST procedure is specified, the event is signal first, then the AST is queued.

In the absents of an event object or an AST procedure, will cause the request to be synchronous. In the case of a synchronous I/O request, the calling thread is not allow to continue until the request completes.

The I/O request completion status is returned in the I/O status block.

ARGUMENTS

channel_id
Supplies the object id of channel to request io on

function_code
Supplies an I/O request function code

iosb
Supplies an I/O status block

completion_event_id
Supplies a user event object to be signaled after I/O the completes

completion_ast
Supplies an ast procedure address to be called when the I/O completes.

ast_parameter
Supplies a parameter for an ast procedure

io_parameters
Supplies a pointer to an I/O parameter record

RETURN VALUES

status$_normal	normal, successful completion
status$_invalid_object_id	invalid object_id
status$_wrong_record_type	Incorrect I/O parameter record for this function code.
status$_object_type_ mismatch	Invalid object
status_wrong_device_class	Invalid function code for this device.
	Interface class specific status

os$synchronize_with_io

(
IN event_id : e$object_id;
BIND iosb : e$iosb;
) RETURNS status;

DESCRIPTION

This service synchronize the calling thread with a currently outstanding asynchronous I/O request.

This service can only be use for asynchronous request that contians at least an event object.

The event object and the IOSB of the previously issued asynchronous I/O request must be supplied as the parameters to this service.

ARGUMENTS

iosb
Supplies an IOSB.

event_id
Supplies an event object ID.

RETURN VALUES

status$_normal	normal, successful completion
status$_invalid_object_id	invalid object id
status$_object_type_ mismatch	invalid object

os$synch_channel_with_fpu

(
IN channel_id : e$object_id;
) RETURNS status;

DESCRIPTION

This routines synchronizes the channel with an FPU object. This is done by copying the sequence number in the FPU object to the channel object.

ARGUMENTS

channel_id
Supplies a object id of the channel object to be synchronized.

RETURN VALUES

status$_normal	normal, successful completion
status$_invalid_object_id	invalid object id
status$_object_type_ mismatch	invalid object

10 Security System Services

os$create_impersonation

(
OUT impersonation_id : e$object_id;
IN object_parameters : e$object_parameters = DEFAULT;
IN remote_nodename : string (*);
IN remote_username : string (*);
IN password : string (*) OPTIONAL;
) RETURNS status;

DESCRIPTION

The os$create_impersonation service allows user mode servers to create an impersonation object. The impersonation object can then be used as input to the os$impersonate_client service to impersonate remote clients.

This service verifies that the remote user is a valid user of the system by requesting the remote user's local user authorization record. If a record exists and the specified password, if any, matches the password in the authorization record, the user is a valid user of the system. If the user is a valid user, the service creates the impersonation object representing the remote user from the remote user's local user authorization record.

The object_parameters parameter is a record consisting of a name, an object container ID, and an ACL. This record, and values for these fields, are optionally provided by the caller. The name field is the name of the object. If a value is not supplied, the object is created without a name. The object container ID field identifies the object container into which the object is inserted, but this field is ignored; the object is inserted into the process-private container. The ACL field supplies additional protection for the object. If a value is not supplied, the object is created without an ACL.

Note: The only server calling this service should be the DFS server.

ARGUMENTS

impersonation_id
Returns the object id of the created impersonation object.

object_parameters
Supplies the object's name, object container, and protection.

remote_nodename
Supplies the name of the remote node.

remote_username
Supplies the name of the remote user.

password
Supplies the password specified by the remote user.

RETURN VALUES

status$_normal	normal, successful completion.
status$_duplicate_object	duplicate object found in object container.
status$_object_container_full	object container full.
status$_invalid_user	the specified user is not authorized to access the system.
status$_invalid_password	the specified password was not valid.

os$create_priv_operation

(
OUT privileged_operation_id : e$object_id;
IN object_parameters : e$object_parameters = DEFAULT;
) RETURNS status;

DESCRIPTION

The os$create_priv_operation creates a privileged operation object. A privileged operation object represents a privileged operation. This object allows software that performs a privileged operation, to determine if a user can perform the privileged operation. If the user has PERFORM_OPERATION access to the privileged operation object, the user is allowed to perform the privileged operation.

Software can have multiple privileged operation objects; the name of each privileged operation object denotes the privileged operation.

The object_parameters parameter is a record consisting of a name, an object container ID, and an ACL. This record, and values for these fields, are optionally provided by the caller. The name field is the name of the object. A value must be supplied because it specifies the name of the privileged operation. The object container ID field identifies the object container into which the object is inserted, but this field is ignored; the object is inserted into the exec$privileged_operation_container system-level container. The ACL field supplies additional protection for the object. If a value is not supplied, the object is created without an ACL.

ARGUMENTS

privileged_operation_id
Returns the object id of the created privileged operation object.

object_parameters
Supplies the object's name, object container, and protection.

RETURN VALUES

status$_normal	normal, successful completion.
status$_duplicate_object	duplicate object found in object container.
status$_object_container_full	object container full.

os$delete_access_control_list

(
IN object_id : e$object_id;
) RETURNS status;

DESCRIPTION

The os$delete_access_control_list services deletes the specified object's access control list.

ARGUMENTS

object_id
Supplies the object id of the object whose ACL is deleted.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.

os$disable_identifier

(
IN identifier : e$identifier;
) RETURNS status;

DESCRIPTION

The os$enable_identifier service disables an identifier in the caller's user identifier list. After the identifier is disabled, it is not used by the system when determining access to objects.

The caller must hold the specified identifier before it can be disabled.

The identifier must have the dynamic attribute in order to be disabled.

ARGUMENTS

identifier
Supplies the identifier to disable.

RETURN VALUES

status$_normal	normal, successful completion.
status$_identifier_not_found	the identifier was not found in the user identifier list.
status$_ident_already_disabled	the identifier was already disabled.
status$_identifier_not_dynamic	the identifier does not have the dynamic attribute.

os$enable_identifier

(
IN identifier : e$identifier;
) RETURNS status;

DESCRIPTION

The os$enable_identifier service enables an identifier in the caller's user identifier list. After the identifier is enabled, it is used by the system when determining access to objects.

The caller must hold the specified identifier before it can be enabled.

The identifier must have the dynamic attribute in order to be enabled.

ARGUMENTS

identifier
Supplies the identifier to enable.

RETURN VALUES

status$_normal	normal, successful completion.
status$_identifier_not_found	the identifier was not found in the user identifier list.
status$_ident_already_enabled	the identifier was already enabled.
status$_identifier_not_dynamic	the identifier does not have the dynamic attribute.

os$get_access_control_list

(
IN object_id : e$object_id;
IN acl : POINTER e$access_control_list;
) RETURNS status;

DESCRIPTION

The os$get_access_control_list service returns the specified object's access control list.

When the service is called, it copies the object's ACL into the ACL pointed to by the ACL parameter. The memory specified by the ACL parameter is managed by the caller and must be large enough to hold the object's ACL. If the ACL is not large enough, the service copies as many entries as the ACL can hold and returns an error status.

ARGUMENTS

object_id
Supplies the object id of the object whose ACL is returned.

acl
Supplies a pointer to the ACL into which a copy of the object's ACL is written. The memory containing the ACL is managed by the caller.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_acl_length_too_small	the size of the specified ACL was not large enough to hold the object's ACL.

os$get_security_monitor

(
OUT security_events_enabled : SET e$security_event [..];
) RETURNS status;

DESCRIPTION

The os$get_security_monitor service returns a summary of the security events that are being monitored.

ARGUMENTS

security_events_enabled
Returns the summary of security events that are being monitored.

RETURN VALUES

status$_normal	normal, successful completion.

os$impersonate_client

(
IN impersonation_id : e$object_id;
IN identifier_option : e$imp_identifier_option;
) RETURNS status;

DESCRIPTION

The os$impersonate_client service allows a server to impersonate a client. A server can restore its own identity by calling the os$restore_server service.

The only context of a client that can be impersonated are the identifiers held by the client. The server can specify to the service how to impersonate the client's identifiers. If the server wants to impersonate the client only, the service sets the caller's identifier list to the list contained in the impersonation object. If the server wants to impersonate the union of the client and the server, the service allocates pool, combines the caller's identifier list and the identifier list in the impersonation object and saves the resultant list in the pool, and sets the caller's identifier list to the list contained in the pool.

Before the service performs the impersonation, it restores the caller's previous identifier list. This allows the caller to impersonate multiple clients in succession without having to make an explicit call to the os$restore_server service.

When a server impersonates a client, the server can access objects as if it were the client.

ARGUMENTS

impersonation_id
Supplies the object id of the impersonation object.

identifier_option
Supplies how the service performs the impersonation. If e$c_client_identifiers value is specified, the service sets the server's identifiers to the client's identifiers in the impersonation object. If the e$c_union_identifiers value is specified, the service combines the server's identifiers with the client's identifiers in the impersonation object.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_object_type_mismatch	the object identified by the imersonation id is not an impersonation object.

os$restore_server

(
);

DESCRIPTION The os$restore_server service restores a server's original identifier list. This service is used by servers that call the os$impersonate_client service to impersonate clients.

ARGUMENTS ***None.***

RETURN VALUES None.

os$set_access_control_list

```
(
IN object_id : e$object_id;
IN acl : POINTER e$access_control_list;
) RETURNS status;
```

DESCRIPTION

The os$set_access_control_list sets the specified object's access control list.

The memory specified by the ACL parameter is managed by the caller. When the service is called, it allocates pool and copies the contents of the specified ACL into the pool.

ARGUMENTS

object_id
Supplies the object id of the object whose ACL is set.

acl
Supplies a pointer to the ACL from which the ACL on the object is set. The memory containing the ACL is managed by the caller.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_invalid_acl	invalid ACL.
status$_invalid_ace	invalid ACE.

os$set_security_monitor

(
IN security_events_enabled : SET e$security_event [..];
IN security_events_disabled : SET e$security_event [..];
) RETURNS status;

DESCRIPTION

The os$set_security_monitor enables or disables the monitoring of security events.

ARGUMENTS

security_events_enabled
Supplies the summary of security events indicating the security events to start monitoring.

security_events_disabled
Supplies the summary of security events indicating the security events to stop monitoring.

RETURN VALUES

status$_normal	normal, successful completion.

os$translate_access_type

(
IN access_type : e$access_type;
IN object_type_name : string (*) OPTIONAL;
OUT access_type_name : string (*);
) RETURNS status;

DESCRIPTION

The os$translate_access_type service translates an access type to its corresponding access type name.

The access type can be either a general or specific access type. If the access type is a general access type, the caller does not have to specify the object_type_name parameter. If the access type is a specific access type, the caller must specify the object_type_name parameter. The object type name denotes the object type that defined the specific access type.

The service performs a case sensitive search to match the object type name.

ARGUMENTS

access_type
Supplies the access type to translate.

object_type_name
Supplies the object type name of the object type that defined the specific access type.

access_type_name
Returns the access type name corresponding to the access type.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_access_type	invalid access type.
status$_invalid_name_length	length of the object type name was not valid.
status$_invalid_object_type	invalid object type specified by the object type name.

os$translate_access_type_name

(
IN access_type_name : string (*);
IN object_type_name : string (*) OPTIONAL;
OUT access_type : e$access_type;
) RETURNS status;

DESCRIPTION

The os$translate_access_type_name service translates an access type name to its corresponding access type.

The access type name can correspond to either a general or specific access type. If the access type name corresponds to a general access type, the caller does not have to specify the object_type_name parameter. If the access type name corresponds to a specific access type, the caller must specify the object_type_name parameter. The object type name denotes the object type that defined the specific access type.

The service performs a case sensitive search to match the access type name and object type name.

ARGUMENTS

access_type_name
Supplies the access type name to translate.

object_type_name
Supplies the object type name of the object type that defined the specific access type.

access_type
Returns the access type corresponding to the access type name.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_name_length	length of the access type name or the object type name was not valid.
status$_invalid_access_type	invalid access type specified by the access type name.
status$_invalid_object_type	invalid object type specified by the object type name.

os$verify_priv_operation

(
IN privileged_operation_id : e$object_id;
) RETURNS status;

DESCRIPTION

The os$verify_priv_operation allows software to determine if a user can perform the privileged operation represented by the specified privileged operation object. If the user has PERFORM_OPERATION access to the privileged operation object, the user is allowed to perform the privieged operation.

ARGUMENTS

privileged_operation_id
Supplies the object id of the privileged operation object.

RETURN VALUES

status$_normal	normal, successful completion.
status$_invalid_object_id	invalid object id.
status$_object_type_ mismatch	the object identified by the privileged operation id is not a privileged operation object.

11 Condition and Exit Handling System Services

os$create_condition_stack

(
IN condition_stack_size : integer[0..];
) RETURNS status;

DESCRIPTION

This system service creates a condition stack of the specified size. If a condition stack already exists, then a new stack is not created and an error status is returned. The stacks size is based on the requested size parameter and is always rounded up two a system defined value. A single guard page is placed at the top of the stack.

ARGUMENTS

condition_stack_size
Supplies the size in bytes for the condition stack being created. This value is always rounded up to an appropriate granularity.

RETURN VALUES

status$_normal	Normal succesful completion of the system service
status$_no_user_stack_va	The condition stack was not created because no virtual address space in the stack region could be found large to staisfy the request.
status$_condition_stack_ exists	A new condition stack was not created since a condition stack already exists.

os$create_last_chance_handler

(
IN condition_handler : e$condition_handler;
OUT handler_id : e$condition_handler_id;
) RETURNS status;

DESCRIPTION

This system service creates a last chance vectored condition handler. Last chance vectored condition handlers are processed in LIFO order during condition delivery. This service places the created last chance handler at the beginning of the last chance vectored condition handler list stored in the calling threads TCR. The service returns a resulting handler_id which may be used to delete a last chance vectored condition handler once it has been created.

The condition handler is linked on the list head in the calling threads TCR indexed by the processor mode that the call was made in.

ARGUMENTS

condition_handler
Supplies the condition handler routine to be invoked when a condition is being dispatched.

handler_id
Returns the handler ID of the created last chance handler. This argument is only valid if the service returns status$_normal.

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessible

os$create_primary_handler

(
IN condition_handler : e$condition_handler;
OUT handler_id : e$condition_handler_id;
) RETURNS status;

DESCRIPTION

This system service creates a primary vectored condition handler. Primary vectored condition handlers are processed in FIFO order during condition delivery. This service places the created primary handler at the end of the primary vectored condition handler list stored in the calling threads TCR. The service returns a resulting handler_id which may be used to delete a primary vectored condition handler once it has been created.

The condition handler is linked on the list head in the calling threads TCR indexed by the processor mode that the call was made in.

ARGUMENTS

condition_handler
Supplies the condition handler routine to be invoked when a condition is being dispatched.

handler_id
Returns the handler ID of the created primary handler. This argument is only valid if the service returns status$_normal.

RETURN VALUES

status$_normal	the service completed without errors
status$_access_violation	a specified parameter is not accessible

os$delete_last_chance_handler

(
IN handler_id : e$condition_handler_id;
) RETURNS status;

DESCRIPTION

This service deletes an existing last chance vectored condition handler. Once deleted, the condition handler will not be called during exception dispatching.

The condition handler is deleted from the list head in the calling threads TCR indexed by the processor mode that the call was made in.

ARGUMENTS

handler_id
Supplies the handler id of the last chance vectored condition handler which is to be deleted.

RETURN VALUES

status$_normal	the service completed without errors
status$_condition_handler_not_found	the last chance vectored condition handler specified by handler_id was not found.

os$delete_primary_handler

(
IN handler_id : e$condition_handler_id;
) RETURNS status;

DESCRIPTION

This service deletes an existing primary vectored condition handler. Once deleted, the condition handler will not be called during exception dispatching.

The condition handler is deleted from the list head in the calling threads TCR indexed by the processor mode that the call was made in.

ARGUMENTS

handler_id
Supplies the handler id of the primary vectored condition handler which is to be deleted.

RETURN VALUES

status$_normal	the service completed without errors
status$_condition_handler_not_found	the primary vectored condition handler specified by handler_id was not found.

12 Miscellaneous System Services

os$get_performance_info

```
(
IN data_list: POINTER e$item_list_type;
IN component_list: POINTER e$item_list_type = NIL;
) RETURNS status;
```

DESCRIPTION

Return requested information about the usage of Mica system resources.

ARGUMENTS

data_list
Supplies the address of an item list which describes the data items to be gathered.

component_list
Supplies the address of the data_list item list. If the data_list specifies data items for a component class, this list specifies the components for which data is to be gathered. If the component item list is not specified, or does not include any components of the requested type, then information is returned for all components of the requested type. If the component_list includes component types for which data is not requested, those component types are ignored.

RETURN VALUES

status$_normal	All data was gathered (success)
status$_no_xxx_component	A specified component of type xxx is missing from the system. Data was returned for all other specified components of that type. (success)
status$_xxx_buffer_overflow	The data buffer for item xxx was not large enough to hold the requested data (failure)
status$_access_violation	The service cannot access the locations specified by one or more items (failure)

os$get_system_information

(
IN system_get_items : POINTER e$item_list_type;
) RETURNS STATUS;

DESCRIPTION The Get System Information system services returns information about the current system.

ARGUMENTS ***system_get_items***
Supplies the item list which specifies the information about the system to return. The following codes are valid:

item code	action
e$c_syi_boottime	Returns the time when the system was booted.
e$c_syi_cpu_type	Returns the CPU processor type.
e$c_syi_software_version	Returns the current version of the operating system.
e$c_syi_number_pagefiles	Returns the current number of pagefiles installed.
e$c_syi_pagefile_free	Returns the total number of free pages in all pagefiles.
e$c_syi_pagefile_used	Returns the total number of used pages in all pagefiles.
e$c_number_of_scalar_cpus	Returns the total number of scalar processors.
e$c_number_of_vector_cpus	Returns the total number of vector processors.
e$c_memory_size	Returns the amount of memory on the system.
e$c_free_page_list_size	Returns the size of the free page list.
e$c_zeroed_page_list_size	Returns the size of the zeroed page list.
e$c_modified_page_list_size	Returns the size of the modifed page list.
e$c_standby_page_list_size	Returns the size of the standby page list.
e$c_bad_page_list_size	Returns the size of the bad page list.

RETURN VALUES

status$_normal	Normal,successful completion.
stauts$_invalid_item_code	error, invalid item code found.

os$get_system_time

```
(
OUT system_time : e$binary_absolute_time;
) RETURNS STATUS;
```

DESCRIPTION The Get System Time service returns the current time in ISO time format.

ARGUMENTS ***system_time***
Returns the current time.

RETURN VALUES

status$_normal	Success, normal completion.
status$_invalid_argument	Error, cannot access argument.

os$get_uid

(
IN desired_number : integer [1..] = 1;
OUT first_uid : e$uid;
OUT number_allocated : integer [0..] OPTIONAL;
) RETURNS STATUS;

DESCRIPTION

The Get UID (Unique Identifier) service returns a UID for use in various components of the Digital Network Architecture.

ARGUMENTS

desired_number
Optionally supplies the desired number of UIDs to allocate. This allows a single call to reserve a group of UIDs for usage. If this argument is not supplied an allocation group of one is returned.

first_uid
Returns the first unique identifier in the allocated group.

number_allocated
Returns the number of UIDs reserved.

RETURN VALUES

status$_normal	Success, normal completion.
status$_invalid_argument	Error, cannot access argument.
status$_not_all_created	Warning, the desired number of UIDs could not be created.

os$install_page_file

(
IN page_file_name : string (*);
) RETURNS STATUS;

DESCRIPTION The Install Page File service installs the specified file as a paging file. The specified file must already exist and not be currently accessed.

ARGUMENTS

page_file_name
Supplies the file name of the specifed page file to install.

RETURN VALUES

status$_normal	Normal, sucessful completion.
file_access_errors	whatever.

os$next_uid

(
IN previous_uid : e$uid;
OUT next_uid : e$uid;
) RETURNS STATUS;

DESCRIPTION

The Next UID (Unique Identifier) service returns a the next UID in a created UID range.

ARGUMENTS

previous_uid
Supplies the previous UID in the range which was returned.

next_uid
Returns the next UID.

RETURN VALUES

status$_normal	Success, normal completion.
status$_invalid_uid	Error, the value for the UID was not a valid UID.

os$set_system_time

(
IN system_time : e$binary_absolute_time;
) RETURNS STATUS;

DESCRIPTION The Set System Time service changes the value of the system time.

ARGUMENTS

system_time
Supplies the new time value for the system time.

RETURN VALUES

status$_normal	Success, normal completion.
status$_invalid_argument	Error, cannot access argument.
status$_no_rights	Error, the thread does not have the proper identifier to change the system time.

A Executive Constants and Data Types

A.1 Executive Constants

```
!
! Executive Defined Constants
!

io$c_deaccess =  -1;          !   (e$request_io,e$execute_io, e$synchronous_io)
io$c_fpu_access =  -2;          !   (e$request_io)
io$c_get_fpu_information =  -3;       !    (e$request_io)
io$c_get_channel_information =  -4;   !    (e$request_io)
io$c_establish_callback =  -5;        !    (e$synchronous_io_call)
io$c_enable_state_change_ast =  -6;   !    (e$request_io, e$synchronous_io_call)
io$c_disable_state_change_ast =  -7;  !    (e$request_io, e$synchronous_io_call)
io$c_item_interface_class =  -1;
io$c_item_fpu_state =  -2;
io$c_item_fpu_bound =  -3;
io$c_item_fp_params_area_size =  -4;
io$c_item_channel_access =  -1;
io$c_item_granted_access =  -2;
io$c_access_request_io :  e$access_type = e$c_specific_access_1;
io$c_access_get_chn_info :  e$access_type = e$c_specific_access_2;
io$c_access_management :  e$access_type = e$c_specific_access_1;
io$c_access_maintenance :  e$access_type = e$c_specific_access_2;
io$c_access_performance :  e$access_type = e$c_specific_access_3;
io$c_access_diagnostic :  e$access_type = e$c_specific_access_4;
io$c_access_allow_channel :  e$access_type = e$c_specific_access_5;
io$c_access_get_fpu_info :  e$access_type = e$c_specific_access_6;
io$c_access_accounting :  e$access_type = e$c_specific_access_7;
io$c_access_access :  e$access_type = e$c_specific_access_8;
io$c_access_fpu_read :  e$access_type = e$c_specific_access_9;
io$c_access_fpu_write :  e$access_type = e$c_specific_access_10;
e$c_es_max_string =  32767;
e$c_max_image_name =  256;
e$c_max_name =  255;
e$c_max_eqvnam_count =  128;
obj$c_max_object_name =  127;       !# This should be 255.
e$c_max_ace_count =  255;
e$c_max_user_name =  32;
k$c_high_priority_level =  63;
k$c_high_processor_number =  31;
e$c_max_ace_identifier_count =  63;
e$c_max_audit_name =  246;          ! Specified by ACL Architecture.
```

A.2 Miscellaneous Data Types

```
!
! Misceleneous Data Types
!
```

```
e$binary_absolute_time :  RECORD
    utc_value :  large_integer;
    inaccuracy :  integer [0..] SIZE (BIT,32);          !!!*** sil limitation should be 48 bits
    reserved :  integer [0..2**16 - 1] SIZE (BIT,16);  !!!*** sil limitation...
    tdf :  integer [ -720.. 780] SIZE (BIT,12);
    version :  integer [0..2**4 - 1] SIZE (BIT,4);
    LAYOUT
        utc_value;
        inaccuracy;
        reserved;
        tdf;
        version;
    END  LAYOUT;
END RECORD;

!
! Unique Identifier Format
!

e$uid :  RECORD
    first_quadword :  large_integer;
    second_quadword :  large_integer;
END RECORD;

!
! Common Item List Format
!

e$item_list_type(ilv_max_entries : integer ) : RECORD
    CAPTURE ilv_max_entries;                                    ! max size number of entries
    ilv_last_inuse_entry :  integer;                             ! index of last valid entry
    ilv_direction :  e$item_list_direction;                      ! direction of entire item list
    ilv_list :  ARRAY[1..ilv_max_entries] OF e$item_list_entry;
END RECORD;

e$item_list_direction :  ( e$c_item_list_in_out,
                            e$c_item_list_in,
                            e$c_item_list_out
                          );

!
! An Item List Consists of an array of item list entries
!

e$item_list_entry :  RECORD
    ile_item_code :  integer;                            ! internal format of an item code
    ile_item_length :  integer;                          ! internal format of an item length
    ile_item_address :  POINTER anytype;                 ! item address
    ile_return_length_address :  POINTER integer;        ! address of return length
    LAYOUT
        ile_item_code ;
        ile_item_length ;
        ile_item_address ;
        ile_return_length_address ;
    END  LAYOUT;
END RECORD;

!
! Common Linked List Entry/Header
!

e$linked_list :  RECORD
    l_flink :  POINTER e$linked_list;
    l_blink :  POINTER e$linked_list;
END RECORD;

!
! Wait Type
!
```

```
e$wait_type :  (
    e$c_wait_any,
    e$c_wait_all
    );

k$processor_mode :  (k$c_kernel, k$c_user);

!
! AST Procedure Format
!

k$normal_ast_routine :
PROCEDURE (
    IN context :  POINTER anytype CONFORM;
    IN system_value :  quadword CONFORM;
          );
```

A.3 I/O Data Types

```
!
! I/O Status Block
!

e$iosb :  RECORD
    condition_value :  longword;                        ! I/O status
    byte_count :  longword;                           ! I/O transfer count
    fp_condition :  quadword;                          ! Filled in by the FP.
END RECORD;

e$fpu_state :  (io$c_fpu_state_offline, io$c_fpu_state_available,
                 io$c_fpu_state_online, io$c_fpu_state_transition,
                 io$c_fpu_state_maintenance);
```

A.4 Logical Name Data Types

```
e$logical_name_list(length : integer [1..]) : RECORD
    CAPTURE length;
    last_valid_entry :  integer;
    context :  large_integer;
    logical_name :  ARRAY [1..length] OF varying_string (e$c_max_name);
    LAYOUT
        length;
        last_valid_entry;
        context;
        logical_name;
    END  LAYOUT;
END RECORD;

e$equivalence_name_list(length : integer [1..e$c_max_eqvnam_count]) : RECORD
    CAPTURE length;
    last_valid_entry :  integer;
    context :  large_integer;
    equivalence_name :  ARRAY [1..length] OF varying_string (e$c_max_name);
    LAYOUT
        length;
        last_valid_entry;
        context;
        equivalence_name;
    END  LAYOUT;
END RECORD;

e$lognam_attributes :  (
    e$c_confine_lognam_attr,
    e$c_noalias_lognam_attr,
    e$c_noshow_lognam_attr
    );
```

A.5 Memory Management Data Types

```
e$page_protections :  (
                       e$c_page_user_read,
                       e$c_page_user_write,
                       e$c_page_user_execute,
                       e$c_page_kernel_read,
                       e$c_page_kernel_write,
                       e$c_page_kernel_execute);

e$mapping_type :  (e$c_data_map, e$c_image_map);

e$page_protection :  SET e$page_protections [..];

e$section_update_flags :  integer; !!!*** fix this
```

A.6 Process Architecture Data Types

```
!
! Process Accounting Summary
!
! The final accounting record contains this information in TLV format
! in addition to fields identifying the process, image name, user ...
!

e$accounting_summary :  RECORD
    acct_total_page_faults :  integer;                  ! Total number of page faults
    acct_hard_page_faults :  integer;                   ! Number of page faults for non resident pages
    acct_soft_page_faults :  integer;                   ! Number of page faults fixed from reclaim lis
    acct_dzro_page_faults :  integer;                   ! Number of demand zero page faults
    acct_com_page_faults :  integer;                    ! Number of copy on modify page faults
    acct_peak_virtual_memory :  integer;                ! Peak virtual memory size
    acct_peak_working_set_size :  integer;              ! Peak working set size
    acct_start_time :  large_integer;                   ! Start time of process
    acct_end_time :  large_integer;                     ! End time of process
    acct_page_file_usage :  integer;                    ! Peak page file usage
    acct_paged_pool_usage :  integer;                   ! Peak paged pool usage
    acct_non_paged_pool_usage :  integer;               ! Peak non paged pool usage
    acct_cpu_and_io :  e$cpu_and_io_summary;            ! CPU and IO accounting summary
END RECORD;

!
! Cpu and IO accounting summary
!
! An instance of this record exists in both the thread control block
! and in the process control block. Updates to the pcb version requires interlocked
! instructions. In the TCB version, only the execute io counters will have to be updated
! using  interlocked instructions
!

e$cpu_and_io_summary :  RECORD
    cis_cpu_cycles :  large_integer;                        ! Number of cycles used by the process or

    !
    ! IO Accounting
    ! Request IO's are counted once.
    ! Each FPU that passes on an IRP (execute_io's) must also record the transfer
    ! by incrementing the counter for its class of FPU
    !

    cis_request_io_count :  integer;                        ! Number of request_io's
    cis_execute_io_count :  ARRAY[e$fpu_class] OF integer;  ! Number of execute_io's per fpu class
END RECORD;

!
! Determines the granularity in the execute io count array
!
```

```
e$fpu_class :  (     e$c_fpu_disk,          ! Disk FPU's
                    e$c_fpu_tape,          ! Tape FPU's
                    e$c_fpu_terminal,      ! Terminal FPU's
                    e$c_fpu_network,       ! Network FPU's
                    e$c_fpu_generic        ! Generic FPU's
                    );

!
! Quota and Resource Usage Data Structures
!

e$quota_vector :  ARRAY[e$quota_types] OF integer;
e$quota_usage :  e$quota_vector;
e$quota_limits :  e$quota_vector;
e$quota_types :  (
                e$c_paging_file_quota,
                e$c_paged_pool_quota,
                e$c_nonpaged_pool_quota,
                e$c_cpu_time_quota
                );


!
! User Job, Process, and Thread Creation Records
!

e$user_record :  RECORD
    user_username :  string(e$c_max_user_name);          ! User Name
    user_security_profile :  e$security_profile;         ! User Security Profile from Authorization Fi
    user_per_user_limits :  e$quota_limits;              ! Per User Resource Limits
    user_per_job_limits :  e$quota_limits;               ! Per Job Resource Limits
    user_per_process_limits :  e$quota_limits;           ! Per Process Resource Limits
    user_thread_priority :  k$combined_priority;         ! Default Thread Priority
    user_thread_affinity :  k$affinity;                  ! Default Thread Affinity
    user_access_restrictions :  e$access_restrictions;   ! Users Access Restrictions
END RECORD;

e$job_record :  RECORD
    job_class :  e$job_class;
    !
    ! Per job Resource limits. This value is used as the
    ! qual_limits value for the job object, and is deducted
    ! from the qual_usage field of the jobs user object.
    ! A value of zero() in any one of fields means to use the
    ! corresponding value of the q_per_job_limit from the
    ! user structure
    !
    job_per_job_limits :  e$quota_limits;
END RECORD;

e$process_record :  RECORD
    process_status_object :  e$object_id;        ! Object ID of processes status object
    process_image_name :  string(e$c_max_image_name); ! Image name for process being created
    !
    ! Per Process Resource limits. This value is used as the
    ! qual_limits value for the process object, and is deducted
    ! from the qual_usage field of the owning job object.
    ! A value of zero() in any one of fields means to use the
    ! corresponding value of the q_per_process_limit from the
    ! user structure
    !
    process_per_process_limits :  e$quota_limits; ! Resource limits for this process
END RECORD;

e$thread_record :  RECORD
    thread_stack_size :  integer;               ! If all 0 then default
    thread_priority :  k$combined_priority;     ! initial thread priority if all 0 then default
    thread_affinity :  k$affinity;              ! complement of affinity If all 0 then all processor
END RECORD;

!
! Misceleneous Thread Creation Parameters
!
```

```
    e$thread_entry_point :  PROCEDURE ();
    k$affinity :  SET integer[0..k$c_high_processor_number];
    k$combined_priority :  integer[0..k$c_high_priority_level];
    k$minor_priority :  integer[0..3];
    e$job_class :  (e$c_jc_invalid,
                    e$c_jc_network,
                    e$c_jc_interactive,
                    e$c_jc_batch,
                    e$c_jc_rsvd1,
                    e$c_jc_rsvd2,
                    e$c_jc_rsvd3,
                    e$c_jc_rsvd4,
                    e$c_jc_rsvd5
                    );

    !
    ! The User Visible Process Control Region
    !

    e$process_control_region :  RECORD
        pcr_image_name :  string(e$c_max_image_name);             ! process image name
        pcr_total_number_of_threads :  integer;                   ! total number of threads for this process
        pcr_number_running_threads :  integer;                    ! number of running threads for this proce
        pcr_object_id :  e$object_id;                             ! process object id -
duplicate of p_obj_id
        pcr_protected_data_hd :  e$linked_list;                   ! List head of protexted data
        pcr_data_block :  POINTER anytype;                        ! Initial process data or NIL
        pcr_data_block_length :  integer;                         ! Length rounded to quad in bytes of data
        pcr_exit_handlers :  e$linked_list;                       ! process level exit handlers
    END RECORD;

    !
    ! The User Visible Thread Control Region
    !

    e$thread_control_region :  RECORD
        tcr_object_id :  e$object_id;                             ! Object ID of this thread
        tcr_stack_array :  ARRAY[0..1] OF e$stack_representation;! tcr stack array
        tcr_current_stack_index :  integer[0..1];                 ! index of current stack
        tcr_pcr_pointer :  POINTER e$process_control_region;      ! Pointer to process control region
        tcr_handler_array :  ARRAY[k$processor_mode] OF e$vectored_handlers; ! vectored handlers for kernel
                                                                              ! user mode
        tcr_exit_handlers :  e$linked_list;                       ! Thread exit handlers User mode only
        tcr_start_address :  e$thread_entry_point;                ! initial start address of thread

        !
        ! Initial Thread Parameters
        !

        tcr_data_block :  POINTER anytype;                        ! Initial thread data or NIL
        tcr_data_block_length :  integer;                         ! Length rounded to quad in bytes
        tcr_parameter1 :  POINTER anytype;                        ! Immediate parameter / or zero()
        tcr_parameter2 :  POINTER anytype;                        ! Immediate parameter / or zero()
        LAYOUT
            tcr_object_id;
            tcr_stack_array;
            tcr_current_stack_index;
            tcr_pcr_pointer;
            tcr_handler_array;
            tcr_exit_handlers;
            tcr_start_address;
            tcr_data_block;
            tcr_data_block_length;
            tcr_parameter1;
            tcr_parameter2;
        END  LAYOUT;
    END RECORD;

    !
    ! Thread Environment Block User Mode R3 points to this
    !
```

```
e$thread_environment_block :  RECORD
    teb_header :  e$common_teb_tcb_header;           ! common teb/tcb header
    teb_vm_zone :  integer;                          ! thread local vm zone
    tls_array_address :  POINTER anytype;            ! address of thread local storage control
    tls_array_free :  integer;                       ! byte offset of first unused tls control array s
    LAYOUT
        teb_header;
        teb_vm_zone;
        tls_array_address;
        tls_array_free;
    END  LAYOUT;
END RECORD;

!
! Misceleneous TCR Constructs
!

e$vectored_handlers :  RECORD
    primary_handlers :  e$linked_list;
    last_chance_handlers :  e$linked_list;
END RECORD;

e$stack_representation :  RECORD
    initial_sp :  POINTER anytype;             ! Initial Value of Condition SP
    stack_limit :  POINTER anytype;            ! Condition Stack Limit
    stack_base :  POINTER anytype;             ! Condition Stack Base
END RECORD;

!
! Common TEB, TCB Header, R3 always points to this structure kernel mode, or user mode
!

e$common_teb_tcb_header :  RECORD
    UNION CASE *
        WHEN 1 THEN                                        ! When teb header first word is length
            teb_length :  integer;                          ! byte length of teb
        WHEN 2 THEN                                        ! When tcb header first word is previous mode
            tcb_previous_mode :  k$processor_mode;          ! saved previous processor mode
    END  UNION;
    tcr_address :  POINTER e$thread_control_region;         ! Pointer to TCR
    LAYOUT
        UNION
            OVERLAY
                teb_length;
            OVERLAY
                tcb_previous_mode;
        END  UNION;
        tcr_address;
    END  LAYOUT;
END RECORD;

!
! Thread performance data
!

e$thread_perf_counters :  RECORD
    tpc_kernel_ticks :  integer;
    tpc_user_ticks :  integer;
    tpc_preemption_switch :  integer;
    tpc_voluntary_switch :  integer;
    tpc_quantum_ends :  integer;
END RECORD;

!
! Item Codes For User, Job, Process, and Thread Services
!
```

```
e$ujpt_item_codes :  ( e$c_ujpt_nil_code,
                        e$c_job_count,
                        e$c_job_ids,
                        e$c_username,
                        e$c_quota_usage,
                        e$c_user_limits,
                        e$c_job_limits,
                        e$c_process_limits,
                        e$c_thread_priority,
                        e$c_thread_affinity,
                        e$c_access_restrictions,
                        e$c_user_id,
                        e$c_process_count,
                        e$c_process_ids,
                        e$c_job_class,
                        e$c_job_id,
                        e$c_parent_id,
                        e$c_sub_process_count,
                        e$c_sub_process_ids,
                        e$c_thread_count,
                        e$c_thread_ids,
                        e$c_process_accounting,
                        e$c_pcr_base,
                        e$c_protected_data,
                        e$c_process_id,
                        e$c_tcr_base,
                        e$c_thread_accounting,
                        e$c_thread_perf_counters,
                        e$c_thread_mnr_priority,
                        e$c_thread_mjr_priority,
                        e$c_get_entire_object
                      );

!
! Exit Status Object Data Types
!

e$status_object_types :  (   e$c_status_process,
                                e$c_status_thread );

e$exit_status_summary :  RECORD
    status_bound_object_type :  e$status_object_types;              ! Process or Thread
    status_bound_object_id :  e$object_id;                          ! Object ID of object reporting st
    status_value :  status;                                         ! Exit Status
    status_string_pointer :  POINTER varying_string(e$c_es_max_string);! Pointer to exit status string
END RECORD;

!
! Get Set information item codes for exit status objects
!

e$exit_status_item_codes :  ( e$c_exit_status_nil_code,
                               e$c_status_value,
                               e$c_status_string,
                               e$c_status_string_set,
                               e$c_status_summary
                             );

e$exit_handler_id :  POINTER anytype;

e$exit_handler_placement :  (
        e$c_beginning_of_list,
        e$c_end_of_list
        );
```

A.7 Object Architecture Data Types

```
!
! All object creation object service routines take as a
! parameter an e$object_parameters record. This record
! specifies the container that the object is to be created in,
! the name of the object, and the acl for the object. Any, or
! all fields can be defaulted to zero() in which case the object
! service routine chooses an appropriate default value.
!

e$object_parameters :  RECORD
    object_container_id :  e$object_id;
    name :  varying_string (obj$c_max_object_name);
    acl :  POINTER e$access_control_list;
END RECORD;

!
! Item codes used in the get information services for
! object architecture defined objects like object containers,
! container directories, and all object headers
!

e$object_item_code :  (
    e$c_acl,
    e$c_allocation_object_id,
    e$c_create_disable,
    e$c_level,
    e$c_logical_name_list,
    e$c_mode,
    e$c_name,
    e$c_nonpaged_pool_charge,
    e$c_object_container_id,
    e$c_object_count,
    e$c_object_id_count,
    e$c_object_id_list,
    e$c_object_state,
    e$c_object_type_name,
    e$c_oid_level,
    e$c_oid_object_container_id,
    e$c_oid_object_id_type,
    e$c_otd_id,
    e$c_owner,
    e$c_paged_pool_charge,
    e$c_pointer_count,
    e$c_principal_object_id,
    e$c_waitable
    );

!
! representation of an object id
!

e$object_id :  QUADWORD;

!
! This data structure is used whenever a variable length list of object
! ids is required
!

e$object_id_list(length : integer [1..]) : RECORD
    CAPTURE length;
    last_valid_entry :  integer;
    context :  large_integer;
    object_id :  ARRAY [1..length] OF e$object_id;
    LAYOUT
        length;
        last_valid_entry;
        context;
        object_id;
    END  LAYOUT;
END RECORD;
```

A.8 Security Related Data Types

```
e$access_control_list(ace_count : integer [0..e$c_max_ace_count]) : RECORD
    CAPTURE ace_count;
    VARIANTS CASE ace_count
        WHEN 0 THEN
            NOTHING;
        WHEN OTHERS THEN
            ace :  ARRAY [1..ace_count] OF e$access_control_entry;
    END  VARIANTS;
    LAYOUT
        ace_count;
        VARIANTS
            OVERLAY
                reserved :  FILLER (longword,*);
                ace;
        END  VARIANTS;
    END  LAYOUT;
END RECORD;

e$access_type :   (
    e$c_general_access_1,
    e$c_general_access_2,
    e$c_general_access_3,
    e$c_general_access_4,
    e$c_general_access_5,
    e$c_general_access_6,
    e$c_general_access_7,
    e$c_general_access_8,
    e$c_general_access_9,
    e$c_general_access_10,
    e$c_general_access_11,
    e$c_general_access_12,
    e$c_general_access_13,
    e$c_general_access_14,
    e$c_general_access_15,
    e$c_general_access_16,
    e$c_general_access_17,
    e$c_general_access_18,
    e$c_general_access_19,
    e$c_general_access_20,
    e$c_general_access_21,
    e$c_general_access_22,
    e$c_general_access_23,
    e$c_general_access_24,
    e$c_general_access_25,
    e$c_general_access_26,
    e$c_general_access_27,
    e$c_general_access_28,
    e$c_general_access_29,
    e$c_general_access_30,
    e$c_general_access_31,
    e$c_general_access_32,
    e$c_specific_access_1,
    e$c_specific_access_2,
    e$c_specific_access_3,
    e$c_specific_access_4,
    e$c_specific_access_5,
    e$c_specific_access_6,
    e$c_specific_access_7,
    e$c_specific_access_8,
    e$c_specific_access_9,
    e$c_specific_access_10,
    e$c_specific_access_11,
    e$c_specific_access_12,
    e$c_specific_access_13,
    e$c_specific_access_14,
    e$c_specific_access_15,
    e$c_specific_access_16,
    e$c_specific_access_17,
        e$c_specific_access_18,
        e$c_specific_access_19,
        e$c_specific_access_20,
        e$c_specific_access_21,
        e$c_specific_access_22,
        e$c_specific_access_23,
        e$c_specific_access_24,
        e$c_specific_access_25,
        e$c_specific_access_26,
        e$c_specific_access_27,
        e$c_specific_access_28,
        e$c_specific_access_29,
        e$c_specific_access_30,
        e$c_specific_access_31,
        e$c_specific_access_32
        );

    e$identifier :  longword;

    e$imp_identifier_option :  (
        e$c_client_identifiers,
        e$c_union_identifiers
        );

    e$security_event :  (
        e$c_acl_audit_security_event
        );

    e$access_ace_flag :  (
        e$c_nonterminal_ace_flag
        );

    e$ace_flag :  (
        e$c_default_ace_flag,
        e$c_nopropagate_ace_flag
        );

    e$ace_type :  (
        e$c_access_ace,
        e$c_audit_ace
        );

    e$audit_ace_flag :  (
        e$c_success_ace_flag,
        e$c_failure_ace_flag,
        e$c_alarm_ace_flag
        );

    e$access_control_entry :  RECORD
        ace_type :  e$ace_type [..] SIZE (byte);
        ace_flags :  SET e$ace_flag [..] SIZE (byte);
        reserved :  byte_data (2);
        UNION CASE *
            WHEN 1 THEN     ! Access ACE specific
                access_flags :  SET e$access_ace_flag [..] SIZE (byte);
                access_identifier_count :  integer [1..e$c_max_ace_identifier_count] SIZE (byte);
                access_access_allowed :  SET e$access_type [..];
                access_identifier :  ARRAY [1..e$c_max_ace_identifier_count] OF e$identifier;
            WHEN 2 THEN     ! Audit ACE specific
                audit_flags :  SET e$audit_ace_flag [..] SIZE (byte);
                audit_access_monitored :  SET e$access_type [..];
                audit_name :  varying_string (e$c_max_audit_name);
        END  UNION;
    END RECORD;
```

A.9 Condition Handling Data Types

```
e$condition_record_pointer :  POINTER e$condition_record;
e$mechanism_record_pointer :  POINTER e$mechanism_record;

e$condition_handler :  PROCEDURE (
    IN condition_record :  e$condition_record_pointer;
    IN mechanism_record :  e$mechanism_record_pointer;
    ) RETURNS status;

e$condition_handler_id :  POINTER anytype;

e$condition_record( argument_number : integer [ 0.. ] ) : RECORD
    CAPTURE argument_number;
    condition_name :  status;
    condition_flags :  SET e$condition_flags [..];
    condition_list :  e$condition_record_pointer;
    processor_status :  arch$processor_status;
    condition_address :  e$instruction_pointer;
    arguments :  ARRAY [ 1..argument_number ] OF e$argument_descriptor;
    LAYOUT
        condition_name;
        condition_flags;
        condition_list;
        processor_status;
        condition_address;
        unused :  FILLER ( longword, 1 );
        argument_number;
        arguments;
    END  LAYOUT;
END RECORD;

e$mechanism_record :  RECORD
    stack_valid :  boolean [ .. ] SIZE ( longword );
    establisher_fp :  e$frame_pointer;
    UNION CASE *
        WHEN 1 THEN
            return_status :  status;
        WHEN 2 THEN
            first_return_register :  arch$register;
            second_return_register :  arch$register;
    END  UNION;

    LAYOUT
        stack_valid;
        establisher_fp;
        UNION
            OVERLAY
                return_status;
            OVERLAY
                first_return_register;
                second_return_register;
        END  UNION;
    END  LAYOUT;
END RECORD;

e$frame_pointer :  POINTER anytype;

arch$processor_status :  integer;  ! dummy definition
arch$register :  longword;
e$instruction_pointer :  POINTER arch$instruction;
arch$instruction :  integer;  ! dummy definition
```

```
e$argument_descriptor :  RECORD
    UNION CASE *
        WHEN 1 THEN
            extent :  integer;
            ptr :  POINTER anytype;
        WHEN 2 THEN
            immediate :  integer;
        WHEN 3 THEN
            large_immediate :  quadword;
    END  UNION;
    class :  integer [0..255] SIZE(byte);
    datatype :  integer [0..255] SIZE(byte);
    size :  integer;
    LAYOUT
        UNION
            OVERLAY
                extent,
                ptr;
            OVERLAY
                immediate;
            OVERLAY
                large_immediate;
        END  UNION;
        class;
        sbz1 :  FILLER(byte,2);
        datatype;
        size;
    END  LAYOUT;
END RECORD;

e$condition_flags :  (
    e$c_condition_unwinding,
    e$c_condition_noncontinuable,
    e$c_condition_exit_unwind,
    e$c_condition_during_ast,
    e$c_condition_async
    );
```

Index