UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T X.753

(10/97)

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

SERIE X: REDES DE DATOS Y COMUNICACIÓN ENTRE SISTEMAS ABIERTOS

Gestión de interconexión de sistemas abiertos – Funciones de gestión y funciones de arquitectura de gestión distribuida abierta

---

Tecnología de la información – Interconexión de sistemas abiertos – Gestión de sistemas: Secuenciador de instrucciones para la gestión de sistemas

Recomendación UIT-T X.753

(Anteriormente Recomendación del CCITT)

RECOMENDACIONES DE LA SERIE X DEL UIT-T

REDES DE DATOS Y COMUNICACIÓN ENTRE SISTEMAS ABIERTOS

REDES PÚBLICAS DE DATOS	
Servicios y facilidades	X.1–X.19
Interfaces	X.20–X.49
Transmisión, señalización y conmutación	X.50–X.89
Aspectos de redes	X.90–X.149
Mantenimiento	X.150–X.179
Disposiciones administrativas	X.180–X.199
INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Modelo y notación	X.200–X.209
Definiciones de los servicios	X.210–X.219
Especificaciones de los protocolos en modo conexión	X.220–X.229
Especificaciones de los protocolos en modo sin conexión	X.230–X.239
Formularios para declaraciones de conformidad de implementación de protocolo	X.240–X.259
Identificación de protocolos	X.260–X.269
Protocolos de seguridad	X.270–X.279
Objetos gestionados de capa	X.280–X.289
Pruebas de conformidad	X.290–X.299
INTERFUNCIONAMIENTO ENTRE REDES	
Generalidades	X.300–X.349
Sistemas de transmisión de datos por satélite	X.350–X.399
SISTEMAS DE TRATAMIENTO DE MENSAJES	X.400–X.499
DIRECTORIO	X.500–X.599
GESTIÓN DE REDES DE INTERCONEXIÓN DE SISTEMAS ABIERTOS Y ASPECTOS DE SISTEMAS	
Gestión de redes	X.600–X.629
Eficacia	X.630–X.639
Calidad de servicio	X.640–X.649
Denominación, direccionamiento y registro	X.650–X.679
Notación de sintaxis abstracta uno	X.680–X.699
GESTIÓN DE INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Marco y arquitectura de la gestión de sistemas	X.700–X.709
Servicio y protocolo de comunicación de gestión	X.710–X.719
Estructura de la información de gestión	X.720–X.729
Funciones de gestión y funciones de arquitectura de gestión distribuida abierta	**X.730–X.799**
SEGURIDAD	X.800–X.849
APLICACIONES DE INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Compromiso, concurrencia y recuperación	X.850–X.859
Procesamiento de transacciones	X.860–X.879
Operaciones a distancia	X.880–X.899
PROCESAMIENTO DISTRIBUIDO ABIERTO	X.900–X.999

Para más información, véase la Lista de Recomendaciones del UIT-T.

NORMA INTERNACIONAL 10164-21

RECOMENDACIÓN UIT-T X.753

TECNOLOGÍA DE LA INFORMACIÓN – INTERCONEXIÓN DE SISTEMAS ABIERTOS – GESTIÓN DE SISTEMAS: SECUENCIADOR DE INSTRUCCIONES PARA LA GESTIÓN DE SISTEMAS

Orígenes

El texto de la Recomendación UIT-T X.753 se aprobó el 24 de octubre de 1997. Su texto se publica también, en forma idéntica, como Norma Internacional ISO/CEI 10164-21.

PREFACIO

La UIT (Unión Internacional de Telecomunicaciones) es el organismo especializado de las Naciones Unidas en el campo de las telecomunicaciones. El UIT-T (Sector de Normalización de las Telecomunicaciones de la UIT) es un órgano permanente de la UIT. Este órgano estudia los aspectos técnicos, de explotación y tarifarios y publica Recomendaciones sobre los mismos, con miras a la normalización de las telecomunicaciones en el plano mundial.

La Conferencia Mundial de Normalización de las Telecomunicaciones (CMNT), que se celebra cada cuatro años, establece los temas que han de estudiar las Comisiones de Estudio del UIT-T, que a su vez producen Recomendaciones sobre dichos temas.

La aprobación de Recomendaciones por los Miembros del UIT-T es el objeto del procedimiento establecido en la Resolución N.° 1 de la CMNT.

En ciertos sectores de la tecnología de la información que corresponden a la esfera de competencia del UIT-T, se preparan las normas necesarias en colaboración con la ISO y la CEI.

NOTA

En esta Recomendación, la expresión "Administración" se utiliza para designar, en forma abreviada, tanto una administración de telecomunicaciones como una empresa de explotación reconocida de telecomunicaciones.

PROPIEDAD INTELECTUAL

La UIT señala a la atención la posibilidad de que la utilización o aplicación de la presente Recomendación suponga el empleo de un derecho de propiedad intelectual reivindicado. La UIT no adopta ninguna posición en cuanto a la demostración, validez o aplicabilidad de los derechos de propiedad intelectual reivindicados, ya sea por los miembros de la UIT o por terceros ajenos al proceso de elaboración de Recomendaciones.

En la fecha de aprobación de la presente Recomendación, la UIT no ha recibido notificación de propiedad intelectual, protegida por patente, que puede ser necesaria para aplicar esta Recomendación. Sin embargo, debe señalarse a los usuarios que puede que esta información no se encuentre totalmente actualizada al respecto, por lo que se les insta encarecidamente a consultar la base de datos sobre patentes de la TSB.

ÍNDICE

Página

Página

NORMA INTERNACIONAL

RECOMENDACIÓN UIT-T

TECNOLOGÍA DE LA INFORMACIÓN – INTERCONEXIÓN DE SISTEMAS ABIERTOS – GESTIÓN DE SISTEMAS: SECUENCIADOR DE INSTRUCCIONES PARA LA GESTIÓN DE SISTEMAS

1 Alcance

La presente Recomendación | Norma Internacional define una función de gestión de sistemas que puede ser utilizada por un proceso de aplicación en un entorno de gestión centralizado o descentralizado con el fin de interactuar a los efectos de la gestión de sistemas, según se define en la Rec. X.700 del CCITT | ISO/CEI 7498-4. La presente Recomendación | Norma Internacional define el secuenciador de instrucciones, que consiste en definiciones genéricas, servicios y unidades funcionales. Esta función está colocada en la capa de aplicación de la Rec. UIT-T X.200 | ISO/CEI 7498-1 y se define de acuerdo con el modelo proporcionado por ISO 9545. El cometido de las funciones de gestión de sistemas se describe en la Rec. UIT-T X.701 | ISO/CEI 10040.

La presente Recomendación | Norma Internacional:

- establece los requisitos de los usuarios con respecto al secuenciador de instrucciones;
- establece modelos que relacionan los servicios prestados por la función con los requisitos de los usuarios;
- define los servicios prestados por la función;
- especifica el protocolo que es necesario para proporcionar los servicios;
- define la relación entre los servicios y las operaciones y notificaciones de información de gestión de sistemas;
- define las relaciones con otras funciones de gestión de sistemas;
- especifica los requisitos de conformidad;
- define un lenguaje de escritura de guiones para utilización en el entorno de secuenciador de instrucciones.

La presente Recomendación | Norma Internacional:

- no define la naturaleza de las implementaciones destinadas a proporcionar el secuenciador de instrucciones;
- no especifica la manera en la cual la gestión es realizada mediante la utilización del secuenciador de instrucciones;
- no define la naturaleza de cualesquiera instrucciones que resulten en la utilización del secuenciador de instrucciones;
- no especifica los servicios necesarios para el establecimiento y la liberación normal y anormal de asociaciones de gestión.

2 Referencias normativas

Las siguientes Recomendaciones y Normas Internacionales contienen disposiciones que, mediante su referencia en este texto, constituyen disposiciones de la presente Recomendación | Norma Internacional. Al efectuar esta publicación, estaban en vigor las ediciones indicadas. Todas las Recomendaciones y Normas son objeto de revisiones, por lo que se preconiza que los participantes en acuerdos basados en la presente Recomendación | Norma Internacional investiguen la posibilidad de aplicar las ediciones más recientes de las Recomendaciones y las Normas citadas a continuación. Los miembros de la CEI y de la ISO mantienen registros de las Normas Internacionales actualmente vigentes. La Oficina de Normalización de las Telecomunicaciones de la UIT mantiene una lista de las Recomendaciones UIT-T actualmente vigentes.

2.1 Recomendaciones CCITT/UIT-T | Normas Internacionales idénticas

– Recomendación UIT-T X.200 (1994) | ISO/CEI 7498-1:1994, *Tecnología de la información – Interconexión de sistemas abiertos – Modelo de referencia básico*: *El modelo básico*.

– Recomendación UIT-T X.210 (1993) | ISO/CEI 10731:1994, *Tecnología de la información – Interconexión de sistemas abiertos – Modelo de referencia básico*: *Convenios para la definición de servicios en la interconexión de sistemas abiertos*.

– Recomendación X.701 del CCITT (1992) | ISO/CEI 10040:1992, *Tecnología de la información – Interconexión de sistemas abiertos – Visión general de la gestión de sistemas.*

– Recomendación UIT-T X.710 (1997) | ISO/CEI 9595:1998, *Tecnología de la información – Interconexión de sistemas abiertos – Servicio común de información de gestión*.

– Recomendación UIT-T X.711 (1997) | ISO/CEI 9596-1:1998, *Tecnología de la información – Interconexión de sistemas abiertos – Protocolo común de información de gestión*: *Especificación*.

– Recomendación X.721 del CCITT (1992) | ISO/CEI 10165-2:1992, *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Definición de la información de gestión.*

– Recomendación X.722 del CCITT (1992) | ISO/CEI 10165-4:1992, *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Directrices para la definición de objetos gestionados.*

– Recomendación UIT-T X.724 (1996) | ISO/CEI 10165-6:1997, *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de la información de gestión: Requisitos y directrices para los formularios de declaración de conformidad de implementación asociados con la gestión de interconexión de sistemas abiertos*.

– Recomendación UIT-T X.725 (1995) | ISO/CEI 10165-7:1996, *Tecnología de la información – Interconexión de sistemas abiertos – Estructura de información de gestión: Modelo general de relación* .

– Recomendación X.730 del CCITT (1992) | ISO/CEI 10164-1:1993, *Tecnología de la información – Interconexión de sistemas abiertos – Gestión de sistemas: Función de gestión de objetos.*

– Recomendación X.731 del CCITT (1992) | ISO/CEI 10164-2:1992, *Tecnología de la información – Interconexión de sistemas abiertos – Gestión de sistemas: Función de gestión de estados.*

– Recomendación X.733 del CCITT (1992) | ISO/CEI 10164-4:1992, *Tecnología de la información – Interconexión de sistemas abiertos – Gestión de sistemas: Función señaladora de alarmas.*

– Recomendación X.734 del CCITT (1992) | ISO/CEI 10164-5:1993, *Tecnología de la información – Interconexión de sistemas abiertos – Gestión de sistemas: Función de gestión de informes de eventos.*

– Recomendación X.735 del CCITT (1992) | ISO/CEI 10164-6:1993, *Tecnología de la información – Interconexión de sistemas abiertos – Gestión de sistemas: Función control de ficheros registro cronológico*.

– Recomendación UIT-T X.739 (1993) | ISO/CEI 10164-11:1994, *Tecnología de la información – Interconexión de sistemas abiertos – Gestión de sistemas: Objetos métricos y atributos*.

– Recomendación UIT-T X.741 (1995) | ISO/CEI 10164-9:1995, *Tecnología de la información – Interconexión de sistemas abiertos – Gestión de sistemas: Objetos y atributos para el control de acceso* .

– Recomendación UIT-T X.746 (1995) | ISO/CEI 10164-15:1995, *Tecnología de la información – Interconexión de sistemas abiertos – Gestión de sistemas: Función de planificación* .

2.2 Pares de Recomendaciones | Normas Internacionales de contenido técnico equivalente

– Recomendación X.209 del CCITT (1988) *Especificación de las reglas básicas de codificación de la notación de sintaxis abstracta uno*.

ISO/CEI 8825:1990, *Information technology – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*.

– Recomendación UIT-T X.291 (1995), *Metodología y marco de las pruebas de conformidad de interconexión de sistemas abiertos de las Recomendaciones sobre los protocolos para aplicaciones del UIT-T – Especificación de sucesiones de pruebas abstractas.*

  ISO/CEI 9646-2:1994, *Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 2: Abstract Test Suite specification*.

– Recomendación UIT-T X.296 (1995), *Metodología y marco de las pruebas de conformidad de interconexión de sistemas abiertos de las Recomendaciones sobre los protocolos para aplicaciones del UIT-T – Declaraciones de conformidad de implementación*.

  ISO/CEI 9646-7:1995, *Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 7: Implementation Conformance Statements*.

– Recomendación X.700 del CCITT (1992), *Marco de gestión para interconexión de sistemas abiertos para aplicaciones del CCITT.*

  ISO/CEI 7498-4:1989, *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 4: Management framework.*

3 Definiciones

A los efectos de la presente Recomendación | Norma Internacional, se aplican las siguientes definiciones.

3.1 Definiciones del modelo de referencia básico

La presente Recomendación | Norma Internacional utiliza los siguientes términos definidos en la Rec. UIT-T X.200 | ISO/CEI 7498-1.

a) sistema abierto;

b) gestión de sistemas.

3.2 Definiciones de convenios de servicio

La presente Recomendación | Norma Internacional utiliza el siguiente término definido en la Rec. UIT-T X.210 | ISO/CEI 10731.

– primitiva.

3.3 Definiciones del marco de gestión

La presente Recomendación | Norma Internacional utiliza los siguientes términos definidos en la Rec. X.700 del CCITT | ISO/CEI 7498-4.

a) información de gestión;

b) objeto gestionado.

3.4 Definiciones de la descripción general de gestión de sistemas

La presente Recomendación | Norma Internacional utiliza los siguientes términos definidos en la Rec. X.701 del CCITT | ISO/CEI 10040.

a) cometido de agente;

b) objeto de soporte de gestión;

c) clase de objeto gestionado;

d) cometido de gestor;

e) notificación;

f) unidad funcional de gestión de sistemas;

g) operaciones de gestión de sistemas.

3.5 Definiciones del servicio común de información de gestión

La presente Recomendación | Norma Internacional utiliza los siguientes términos definidos en la Rec. UIT-T X.710 | ISO/CEI 9595.

a) atributo;

b) servicio común de información de gestión.

3.6 Definiciones adicionales

En la presente Recomendación | Norma Internacional se definen los siguientes términos.

3.6.1 secuenciador de instrucciones: Un objeto de soporte de gestión que representa un recurso que funciona con un cometido de gestor como un destino de notificación y como un iniciador de operaciones determinado por sus guiones de lanzamiento, con la capacidad de delegar actividades de gestión.

3.6.2 guión de lanzamiento: Un objeto gestionado que representa las instrucciones que han de ser ejecutadas por un secuenciador de instrucciones.

3.6.3 hilo: Objeto gestionado que representa la ejecución de un guión de lanzamiento. Los resultados o errores de las ejecuciones del guión de lanzamiento son devueltos por el hilo.

3.6.4 hilo suspendible: El hilo suspendible se deriva de la clase de objeto gestionado hilo. Estos hilos son generados por lanzadores asíncronos. Pueden ser suspendidos por medio de la acción suspender dirigida a ellos y reanudados por medio de la acción reanudar dirigida a ellos.

3.6.5 lanzador: Un objeto de soporte de gestión al cual se puede dirigir una activación para iniciar la ejecución de un guión de lanzamiento. Un lanzador sirve como un objeto gestionado de valor inicial para un hilo.

3.6.6 lanzador asíncrono: Un lanzador asíncrono se deriva del lanzador. Devuelve una notificación de resultado de activación sin esperar los resultados de la ejecución de los guiones de lanzamiento. Los resultados o errores de las ejecuciones del guión de lanzamiento son notificados directamente a partir del hilo.

3.6.7 lanzador síncrono: Un lanzador síncrono se deriva del lanzador. Devuelve la notificación del resultado de activación o la alarma de error de procesamiento después que obtiene todos los resultados y errores de la ejecución de los hilos una vez que éstos completan su ejecución.

3.6.8 activador: Un iniciador de ejecución de guiones que hace que un lanzador genere uno o más hilos. Dirige una instrucción a un lanzador, en forma de organizador, operaciones, notificaciones o acción local.

3.6.9 instrucción: Instrucción destinada a una actividad de gestión que se realiza en el sistema agente conforme al contenido de un guión de lanzador. Se describe mediante un lenguaje de escritura de guiones. Actualmente, el lenguaje de escritura de guiones de gestión de sistemas se define en el anexo F.

4 Abreviaturas

A los efectos de la presente Recomendación | Norma Internacional, se aplican las siguientes abreviaturas:

ASN.1 Notación de sintaxis abstracta uno (*abstract syntax notation one*)

CMIS Servicio común de información de gestión (*common management information service*)

CS Secuenciador de instrucciones (*command sequencer*)

IVMO Objeto gestionado de valor inicial (*initial value managed object*)

OSI Interconexión de sistemas abiertos (*open systems interconnection*)

LP Lanzador (*launch pad*)

SMSL Lenguaje de escritura de guiones de gestión de sistemas (*systems management scripting language*)

5 Convenios

La presente Recomendación define servicios para el secuenciador de instrucciones que siguen los convenios descriptivos definidos en la Rec. UIT-T X.210 | ISO/CEI 10731. En la cláusula 9, la definición de cada servicio incluye un cuadro que enumera los parámetros de servicio. Para una primitiva de servicio dada, la presencia de cada parámetro se describe mediante uno de los valores siguientes:

M El parámetro es obligatorio (mandatory).

(=) El valor del parámetro es igual al valor del parámetro en la columna a la izquierda.

U La utilización del parámetro es una opción del usuario del servicio.

– El parámetro no está presente en la interacción descrita por la primitiva en cuestión.

C El parámetro es condicional. Las condiciones son definidas por la prueba que describe este parámetro.

P El parámetro está sujeto a las restricciones impuestas por la Rec. UIT-T X.710 | ISO/CEI 9595.

NOTA – Los parámetros marcados con una "P" en los cuadros de servicios de esta Especificación corresponden directamente con los parámetros del CMIS, sin cambiar la semántica ni la sintaxis de los parámetros.

El tipo de carácter utilizado para las directrices para las directrices para la definición de objetos gestionados (GDMO, *guidelines for definition of managed objects*), ASN.1 y el modelo de relación general (GRM, *general relationship model*) en la presente Recomendación | Norma Internacional es Courier. El BNF para SMSL en F.14.11 es Courier New. En los anexos F y G, los parámetros de funciones del SMSL aparecen en cursivas.

6 Requisitos

Los requisitos que se han de satisfacer son:

- Requisitos de usuario:
  - Permitir la delegación de actividades de gestión.
  - Reducir la cantidad de comunicación que debe producirse entre el gestor y los agentes.
  - Permitir que funcionen sistemas de gestor delegados en sistemas de agente incluso cuando las comunicaciones entre un gestor y los sistemas de agente han sido interrumpidas o no son posibles.
  - Proporcionar un control flexible de las actividades de gestión.
  - Proporcionar un lenguaje de guión que puede describir los procedimientos para ejecutar operaciones de gestión.
  - Permitir que los sistemas delegados ejecuten operaciones CMIS en secuencia.
- Requisitos operacionales:
  - Ejecución preplanificada o retardada de una operación de gestión de sistemas.
  - Capacidades para modificar la petición de una ejecución preplanificada o retardada.
  - Capacidades para iniciar, suspender, reanudar y terminar operaciones de gestión de sistemas basadas en acciones de gestión temporal o en la ocurrencia de eventos.
  - Capacidades para informar y registrar el resultado de ejecuciones preplanificadas o retardadas.
  - Capacidad para enviar notificaciones cuando se producen cambios de estados.

7 Modelos

7.1 Descripción de modelo

El modelo describe cómo la ejecución activada, preplanificada o retardada de operaciones de gestión de sistemas pueden ser ejecutadas por el secuenciador de instrucciones. Describe los componentes conceptuales, la relación entre estos componentes y los estados y posibles transiciones de estado.

La figura 1 es una descripción esquemática de la capacidad del secuenciador de instrucciones de un sistema.

La funcionalidad de un secuenciador de instrucciones es modelada por los objetos guión de lanzamiento, hilo y lanzador. Es una abstracción de OSI de la ejecución de operaciones preplanificadas o retardadas en sistemas abiertos. Un secuenciador de instrucciones puede contener cualquier número de lanzadores, para los que el secuenciador de instrucciones sirve de proveedor de servicio. Cada lanzador puede ejecutar un guión de lanzamiento a la vez, o varios a la vez. Al recibir una orden de activación, el lanzador inicia la ejecución de un guión de lanzamiento. Además del componente identificador de activador, éste puede especificar un nombre de guión de lanzamiento (identificador de guión) y argumentos de entrada al guión como parámetros dentro del componente lista de parámetros de ejecución.

Hay dos tipos de lanzadores, lanzador asíncrono y lanzador síncrono. El lanzador asíncrono devuelve una notificación de resultado de activación sin esperar los resultados de la ejecución de los guiones de lanzamiento. Los resultados o errores de las ejecuciones de guiones de lanzamiento son notificados directamente por el hilo. Por otra parte, un lanzador síncrono devuelve notificación de resultado de activación o alarma de error de procesamiento después que obtiene todos los resultados y errores de ejecución de los hilos, cuando éstos completan su ejecución.

Un caso de guión de lanzamiento puede contener cualquier número de instrucciones individuales. El componente lista de parámetros de ejecución del lanzador es una lista de guiones (identificados por sus identificadores de guión) que se han de ejecutar y los parámetros de entrada correspondientes necesarios para ejecutar estos guiones. Se puede especificar una lista de parámetros de ejecución por defecto para que un lanzador la ejecute si recibe una activación en la cual no están especificados los parámetros de activación. Si el lanzador no está configurado para ejecutar una lista de parámetros de ejecución por defecto y el componente lista de parámetros de ejecución no es suministrado por el activador y si el lanzador recibe un intento del activador de activarlo, se devuelve un código de error "no hay guión" en el campo código de error de la notificación de resultado de activación. El lanzador tiene un atributo lista de guiones disponibles que puede ser configurado para identificar guiones que pueden ser ejecutados por él. Si un componente lista de parámetros de ejecución está presente en la activación, el lanzador verifica si cada identificador de guión del componente de parámetros de ejecución está presente en su atributo lista de guiones disponibles. Sólo se ejecutan los guiones indicados por el identificador de guión que están presentes en el atributo lista de guiones disponibles. Si ninguno de los identificadores de guión está presente en la lista de guiones disponibles, el lanzador devuelve un código de error “guión rechazado” en los resultados de la activación y no se produce la ejecución del guión.

Las clases de objeto lenguaje de escritura de guiones especializados se derivan de la clase de objeto guión de lanzamiento. En consecuencia, estas instrucciones pueden ser especificadas como casos de guiones especializados. Los conjuntos múltiples instrucciones de guiones de lanzamiento pueden ser ejecutadas secuencialmente o en paralelo por los hilos, según el tipo de datos parámetro de ejecución. Varios niveles jerarquizados de subhilos pueden ser necesarios para ejecutar guiones.

Para iniciar el comportamiento de ejecución de guiones de lanzamiento, se puede dirigir una activación al objeto lanzador. Los activadores que no están parametrizados pueden activar el lanzador cuando éste tiene una lista de parámetros de ejecución por defecto.

El lanzador actúa como un IVMO para un hilo y suministra la lista de parámetros de ejecución del atributo parámetros ejecutores del hilo. La lista de parámetros de ejecución puede ser un parámetro de ejecución, una secuencia de parámetros de ejecución o un conjunto de parámetros de ejecución. El parámetro de ejecución es una secuencia de identificadores de guiones y de parámetros de guiones. El identificador de guión identifica el nombre del objeto gestionado del objeto escritura de guión que se ha de ejecutar y los parámetros de guión suministran los valores de parámetro que se necesitan como entradas al objeto escritura de guión. Si se especifica una secuencia de parámetros de ejecución, el lanzador genera un hilo para ejecutar los guiones y suministra al hilo el identificador de guión y los parámetros de guión (si procede) a partir de los parámetros de ejecución. Al completarse el hilo, esto se repite para el resto de los identificadores de guión en la lista en secuencia. Si se especifica un conjunto de parámetros de ejecución, el lanzador suministra el conjunto de identificadores de guión y de parámetros de guión (si procede) a los hilos y se ejecutan los correspondientes guiones en paralelo. La semántica de los parámetros que pasa entre el lanzador y los hilos depende del mecanismo de paso de parámetros sustentado por el lenguaje de guiones en el cual está escrito el guión.

Se asigna un hilo a la ejecución de un guión. Este hilo puede generar otros, si es necesario. Esto puede ocurrir cuando un guión invoca otros guiones. En este caso, el hilo que ejecuta un guión llamante genera un subhilo, que transfiere el identificador de guión y los parámetros de guión (si procede) del guión llamado al subhilo. La semántica de paso de parámetros entre hilos y subhilos depende del mecanismo de paso de parámetros sustentado por el lenguaje de escritura de guión en el cual está escrito el guión.

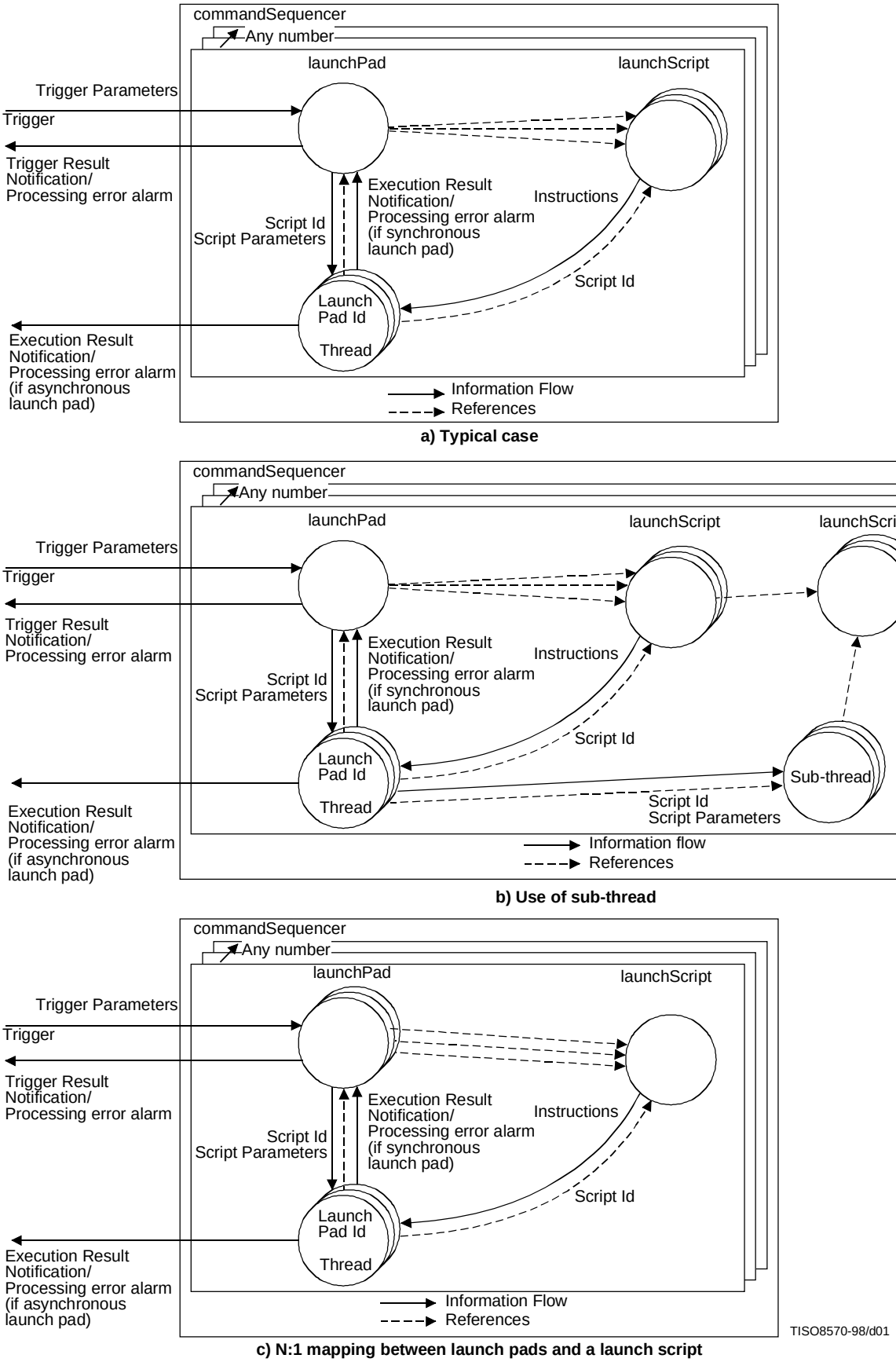


Figura 1 – Modelo de secuenciador de instrucciones

Los lanzadores asíncronos deben generar hilos suspendibles. Un hilo suspendible puede ser suspendido y reanudado por medio de las acciones suspender y reanudar, respectivamente. Los hilos generados por lanzadores síncronos no pueden ser suspendidos y reanudados. Todos los hilos relacionados con la ejecución de un guión pueden ser suspendidos o reanudados en ambos casos.

Un hilo se ha completado después que todos sus subhilos se han completado satisfactoriamente o han informado un error. Cuando sucede esto, se ha completado la ejecución de un guión de lanzamiento; el lanzador correspondiente vuelve a un estado inactivo (reposo). El hilo está contenido por el objeto que lo genera. Un hilo puede estar contenido en un lanzador o en otro hilo.

Múltiples lanzadores pueden hacer referencia a un guión de lanzamiento determinado. Múltiples hilos pueden hacer referencia a ese guión de lanzamiento. La existencia de un guión es independiente de las referencias hechas por lanzadores o hilos. Los guiones de lanzamiento se definen en las clases de guiones especializados derivadas de la clase de objeto guión de lanzamiento. La semántica y la sintaxis de estos guiones se especifican en la definición del lenguaje de escritura de guiones en el cual están escritos los guiones. La definición del lenguaje de escritura de guiones especifica también un conjunto de funciones básicas de escritura de guiones que son necesarias para proporcionar la capacidad de control y de procesamiento a los guiones de lanzamiento.

La clase de objeto gestionado guión de cadena general se debe utilizar para escribir guiones que se representan en forma de una cadena general. El atributo nombre de lenguaje de guión indica el nombre de ese lenguaje de escritura de guiones y el atributo contenido de guión representa el guión escrito en este lenguaje de escritura de guión en forma de una cadena general. Los anexos F y G definen un lenguaje de escritura de guiones especializados, el lenguaje de escritura de guión de gestión de sistema (SMLS), como el lenguaje de escritura de guión en el cual se deben escribir los guiones representados en la forma de una cadena general. Es posible que se definan otras clases de guiones. El anexo D define un lenguaje de escritura de guión, cmisScript, en forma de objetos gestionados que pueden ser utilizados para escribir guiones en el entorno CMIS.

7.2 Proceso de activación e informe de resultados

Los activadores dirigidos al lanzador pueden tener diversas formas, tales como planificadores, operaciones, notificaciones y acción local. Cuando un lanzador recibe una activación, genera uno o más hilos para ejecutar un guión. Después que todos los hilos relacionados con una activación han sido generados, un lanzador asíncrono emite una notificación de resultado de activación que incluye conjuntos de identificadores de hilo y de identificadores de guión. Los resultados de la ejecución del guión son propagados por los hilos como notificaciones de resultado de ejecución directamente al gestor en el caso del lanzador asíncrono. Una vez completados todos los hilos relacionados con una activación, un lanzador síncrono sincroniza todos los resultados o errores de ejecución de los hilos y emite al gestor una notificación de resultado de activación que incluye conjuntos de identificadores de hilos, identificadores de guión y resultados o errores de ejecución.

El atributo tipo de resultado de ejecución del guión identifica el tipo de resultado esperado de la ejecución del guión y debe corresponder con el atributo tipo de resultado de ejecución del resultado de la ejecución. El campo código de error del resultado de la ejecución se pone al código "ningún error" cuando la ejecución tiene éxito y, en los demás casos, se pone al código de error apropiado.

Un hilo ejecutor puede ser terminado espontáneamente al completar su ejecución o en condiciones anormales. En el segundo caso, el hilo indica terminación anormal emitiendo una notificación de alarma de error de procesamiento.

Las notificaciones de resultado de la ejecución y de alarma de error de procesamiento son emitidas por el hilo y enviadas al destino o destinos de notificación apropiados. En el caso de un lanzador asíncrono, estas notificaciones son enviadas a destinos de notificación externos mientras que en el caso de un lanzador síncrono, estas notificaciones se envían al lanzador.

Un gestor puede terminar voluntariamente todos los procesos de lanzamiento por medio de una operación suprimir dirigida al lanzador correspondiente. Al recibir una operación suprimir, si la vinculación de nombre hilo-lanzador (thread-launchPad) incluye la definición "SUPRIMIR-SUPRIME-OBJETOS-CONTENIDOS" (DELETE DELETES-CONTAINED-OBJECTS), todos sus hilos que ejecutan el guión son terminados y suprimidos. El lanzador es suprimido después.

Un gestor puede terminar voluntariamente todas las ejecuciones relacionadas con un hilo por medio de una operación suprimir dirigida al lanzador correspondiente. Al recibir una operación suprimir, si la vinculación de nombre hilo-hilo (thread-thread) incluye la definición "SUPRIMIR-SUPRIME-OBJETOS-CONTENIDOS", todos sus subhilos que ejecutan el guión son terminados y suprimidos. El lanzador es suprimido después.

Para terminar la ejecución de todos los guiones que están siendo ejecutados por un lanzador síncrono o asíncrono, se puede dirigir una acción terminar al lanzador. Todos los hilos relacionados con la ejecución de guiones son terminados y suprimidos cuando una acción terminar es recibida por el lanzador.

El lanzamiento de todos los hilos que están siendo ejecutados por un lanzador síncrono o asíncrono puede ser suspendido por una acción suspender dirigida al lanzador y reanudado después por una acción reanudar.

La ejecución de guiones por hilos suspendibles generados por un lanzador asíncrono puede ser suspendida por una acción suspender dirigida al hilo y después reanudada por una acción reanudar. Se debe suministrar el identificador de hilo, devuelto en la notificación de resultado de la activación, como un parámetro para las acciones suspender y reanudar.

El lanzador tiene atributos para supervisar un atributo específico de un objeto específico. Si se cambia el valor del atributo supervisado, se genera una activación para ejecutar una lista de guiones especificados.

Si el atributo supervisado es un contador de discriminación de eventos (EDC, *event discrimination counter*) definido en el anexo C, las notificaciones filtradas por el EDC, activan el lanzamiento de guiones por el lanzador.

7.3 Gestión del secuenciador de instrucciones

Los valores de atributo de los objetos gestionados lanzador, hilo, guión de lanzamiento y escritura de guiones especializados son extraídos y modificados mediante las operaciones obtención y fijación, respectivamente.

Los cuadros 1 a 5 muestran la correspondencia de los atributos de estado de los objetos gestionados secuenciador de instrucciones, lanzador, hilo y guión con los estados definidos en la Rec. X.731 del CCITT | ISO/CEI 10164-2.

NOTA – "–" significa cualquier valor.

Cuadro 1 – Estados de secuenciador de instrucciones

Estado de secuenciador de instrucciones	Estado administrativo	Estado operacional
CS no operacional	–	Inhabilitado
CS operacional	Desbloqueado	Habilitado
CS bloqueado	Bloqueado	Habilitado
CS parado	Parado	Habilitado

Cuando un secuenciador de instrucciones tiene inhabilitado el estado operacional, está en un estado totalmente inoperable y sus lanzadores no ejecutan guiones. Si está en un estado habilitado, un evento que consiste en la ejecución de una operación en la frontera del objeto gestionado puede hacer que pase de un estado administrativo bloqueado a un estado desbloqueado o viceversa. Cuando el secuenciador de instrucciones pasa al estado bloqueado, hace que sus lanzadores suspendan la ejecución de guiones. Alternativamente, cuando pasa a un estado administrativo desbloqueado, los lanzadores están disponibles para comenzar o reanudar la ejecución de guiones de lanzamiento.

Cuando un lanzador tiene un estado operacional inhabilitado, está en un estado totalmente inoperable y no puede ejecutar guiones. Si está en un estado habilitado, un evento que consiste en la ejecución de una operación en las fronteras del objeto gestionado puede hacerlo pasar de un estado administrativo bloqueado a un estado desbloqueado o viceversa. Cuando el lanzador pasa al estado bloqueado, suspende la ejecución de los guiones. Alternativamente, cuando pasa a un estado administrativo desbloqueado, los lanzadores están disponibles para comenzar o reanudar la ejecución de guiones de lanzamiento. El lanzador pasa a estar inactivo mediante un proceso de control interno de acuerdo con un horario predeterminado y su valor de estado de disponibilidad es fuera de servicio. Una acción suspender hace que el estado de control cambie a suspendido y una acción reanudar hace que vuelva a su valor por defecto, vacío.

Cuadro 2 – Estados de lanzador

Estado de lanzador	Estado administrativo	Estado operacional	Estado de control	Estado de utilización	Estado de disponibilidad
LP no operacional	–	Inhabilitado	–	–	
LP operacional	Desbloqueado	Habilitado	–	Ocupado	
LP operacional	Desbloqueado	Habilitado	–	Reposo	–
LP bloqueado	Bloqueado	Habilitado	–	Reposo	–
LP en servicio	–	–	–	–	No está fuera de servicio
LP fuera de servicio	–	–	–	–	Fuera de servicio
LP suspendido	–	–	Suspendido	–	–
LP reanudado	–	–	Vacío	–	–

Cuadro 3 – Estados de hilo

Estado de hilo	Estado operacional
Hilo no operacional	Inhabilitado
Hilo operacional	Habilitado

El hilo está en un estado habilitado cuando ejecuta un guión y en un estado inhabilitado cuando no lo ejecuta.

Cuadro 4 – Estados de hilo suspendible

Estado de hilo suspendible	Estado operacional	Estado operacional
Hilo suspendible no operacional	Inhabilitado	–
Hilo suspendible operacional	Habilitado	–
Hilo suspendible suspendido	–	Suspendido
Hilo suspendible reanudado	–	Vacío

Una acción suspender hace que el estado de control del hilo suspendible cambie a suspendido y una acción reanudación hace que vuelva a su valor por defecto, vacío.

Cuadro 5 – Estados de guión de lanzamiento

Estado de guión de lanzamiento	Estado administrativo
Ejecución LS autorizada	Desbloqueado
Ejecución LS no autorizada	Bloqueado

Un evento que consiste en una operación ejecutada en la frontera del objeto gestionado puede hacer que un guión pase de un estado administrativo bloqueado a un estado desbloqueado o viceversa. Cuando el guión de lanzamiento pasa al estado bloqueado, no puede ser ejecutado por un lanzador distinto a los que están actualmente ejecutándolo. Alternativamente, cuando pasa a un estado administrativo desbloqueado, el guión de lanzamiento está disponible para ser ejecutado por otros lanzadores.

7.4 Planificación del secuenciador de instrucciones

Un lote de planificación de un planificador externo proporciona la capacidad de planificar la activación de lanzadores del secuenciador de instrucciones por los activadores. El atributo de estado de disponibilidad del lanzador será cambiado de "fuera de servicio" a "no está fuera de servicio" de acuerdo con las características de planificación especificadas por el objeto gestionado planificador externo. La semántica del lote de planificación del planificador externo se describe en la Rec. X.734 del CCITT | ISO/CEI 10164-5 y en la Rec. X.735 del CCITT | ISO/CEI 10164-6.

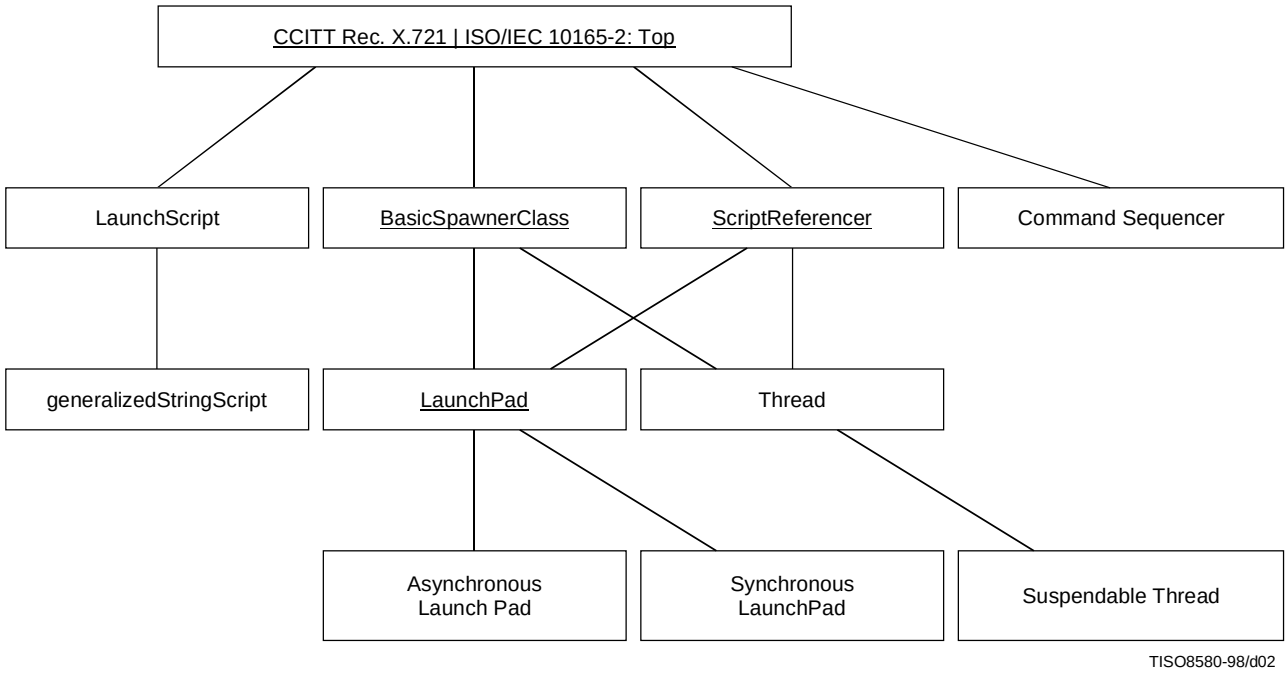
7.5 Control de acceso

El secuenciador de instrucciones debe permitir el acceso a todos los objetos gestionados que son puestos en funcionamiento por sus hilos. El comportamiento del hilo cuando se rechaza una operación de un objeto debe estar definido en el guión. Por ejemplo, si se rechaza una operación, un error de intento de acceso no autorizado puede ser devuelto por una notificación de alarma de error de procesamiento.

8 Definiciones genéricas

8.1 Objetos gestionados

Esta Especificación define un conjunto de clases de objetos gestionados. La estructura de herencia de estas clases de objetos gestionados se muestra en la figura 2.



NOTA – Se subrayan las clases de objeto que no se pueden ejemplificar.

Figura 2 – Estructura de herencia de recursos del secuenciador de instrucciones

La estructura de contenimiento de estas clases de objetos gestionados se muestra en la figura 3.

8.1.1 Secuenciador de instrucciones

8.1.1.1 Visión general

- Un objeto de soporte de gestión que representa un recurso que actúa con un cometido de gestor como un invocador de operaciones determinado por sus guiones de lanzamiento y como un destino de notificación.
- Actúa como proveedor de servicio para un lanzador.
- Contiene uno o más lanzadores.

8.1.1.2 Lotes del objeto soporte de gestión de secuenciador de instrucciones

El objeto soporte de gestión de secuenciador de instrucciones tiene el siguiente lote obligatorio:

- lote de secuenciador de instrucciones (command sequencer package).

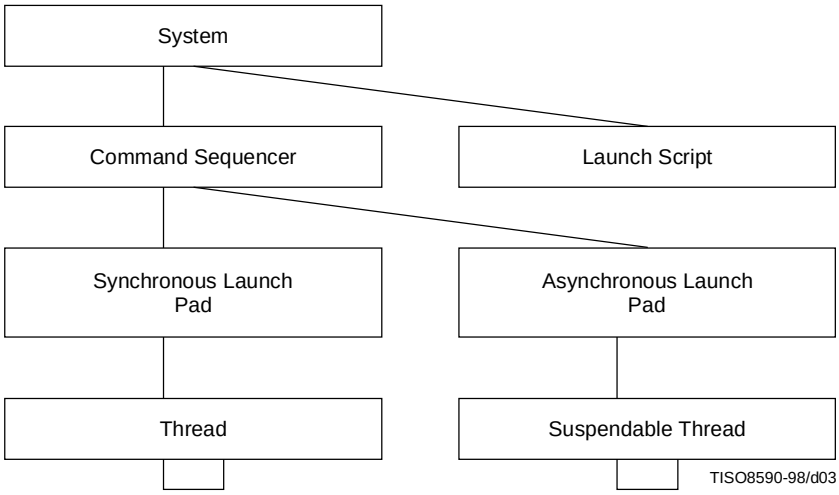


Figura 3 – Estructura de contenimiento de recursos del secuenciador de instrucciones

8.1.1.3 Características del secuenciador de instrucciones

La clase de objeto soporte de gestión de secuenciador de instrucciones tiene los siguientes atributos.

8.1.1.3.1 Identificador de secuenciador de instrucciones

El valor de este atributo define un caso de la clase de objeto soporte de gestión de secuenciador de instrucciones.

8.1.1.3.2 Estado administrativo

Este atributo representa la capacidad administrativa del secuenciador de instrucciones para ejecutar su función. Se definen los siguientes estados administrativos:

a) Desbloqueado – Los lanzadores del secuenciador de instrucciones están autorizados a comenzar o reanudar la ejecución de guiones de lanzamiento.

b) Bloqueado – Los lanzadores del secuenciador de instrucciones no están autorizados a comenzar la ejecución de guiones de lanzamiento. Si hay ejecuciones en curso, se suspenden.

c) Cierre – El secuenciador de instrucciones está parado y sus lanzadores no están autorizados a ejecutar ningún guión.

8.1.1.3.3 Estado operacional

Este atributo representa la capacidad operacional del secuenciador de instrucciones de ejecutar sus funciones.

Se definen los siguientes estados operacionales:

a) Habilitado – El secuenciador de instrucciones es operacional y está listo para utilización.

b) Inhabilitado – El secuenciador de instrucciones no está disponible para utilización.

8.1.1.4 Notificaciones del secuenciador de instrucciones

La clase de objeto soporte de gestión de secuenciador de instrucciones tiene las siguientes notificaciones:

– Creación de objeto, que se define en la Rec. X.730 del CCITT | ISO/CEI 10164-1.

– Supresión de objeto, que se define en la Rec. X.730 del CCITT | ISO/CEI 10164-1.

– Cambio de estado, que se define en la Rec. X.731 del CCITT | ISO/CEI 10164-1.

8.1.2 Hilo

8.1.2.1 Visión general

– Un objeto de gestión que modela la ejecución de instrucciones y ejecuta instrucciones de acuerdo con un guión.

– Subclase de las clases de objetos gestionados generador básico y referenciador de guiones.

– Cada hilo está dedicado a un lanzador síncrono.

– Esta clase de objeto tiene una notificación que notifica el resultado de la ejecución del guión de lanzamiento.

8.1.2.2 Lotes de la clase de objeto gestionado hilo

La clase de objeto gestionado hilo tiene los siguientes lotes obligatorios:

- lote de hilos (thread package);
- lote de resultado de ejecución (execution result package).

8.1.2.3 Características de la clase de objeto gestionado hilo

La clase hilo tiene los siguientes atributos.

8.1.2.3.1 Identificador de hilo

El valor de este atributo identifica un caso de la clase de objeto gestionado hilo.

8.1.2.3.2 Identificador de guión

El valor de este atributo identifica un caso de la clase de objeto gestionado guión de lanzamiento que se está ejecutando.

8.1.2.3.3 Parámetros de ejecución

El componente parámetros de guión de este atributo es una lista de valores de parámetros suministrados por un lanzador, que necesita ejecutar un guión. El componente identificador de guión de este atributo indica el guión o guiones que deben ser ejecutados con los parámetros correspondientes.

8.1.2.3.4 Estado operacional

Este atributo representa la capacidad operacional del hilo de ejecutar sus funciones. Se definen los siguientes estados operacionales:

a) Habilitado – El hilo es operacional y está ejecutando un guión.

b) Inhabilitado – El hilo no es operacional.

8.1.2.4 Notificaciones de la clase de objeto gestionado hilo

La clase de objeto gestionado hilo tiene las siguientes notificaciones, que envía a la clase de objeto gestionado lanzador:

- resultado de ejecución, según se define en 8.2.1;
- alarma de error de procesamiento según se define en la Rec. X.733 del CCITT | ISO/CEI 10164-4.

8.1.3 Hilo suspendible

8.1.3.1 Visión general

- Subclase de la clase de objeto gestionado hilo.
- Generado por un lanzador asíncrono.
- Puede ser suspendido y reanudado por medio de las acciones suspender y reanudar.

8.1.3.2 Lotes del hilo suspendible

Los lotes obligatorios de la clase de objetos gestionados hilos suspendibles son:

- Aceptador de suspensión y reanudación (suspend resume accepter).

8.1.3.3 Características del hilo suspendible

El hilo suspendible tiene los siguientes atributos.

8.1.3.3.1 Estado de control

El atributo estado de control se define en la Rec. X.731 del CCITT | ISO/CEI 10164-2. El valor por defecto es vacío. Si se suspende la ejecución, el estado de control cambia a "suspendido" y si se reanuda la ejecución, el valor cambia a vacío.

8.1.3.4 Acciones de la clase de objeto gestionado hilo suspendible

Las siguientes acciones pueden ser dirigidas a la clase de objeto gestionado hilo suspendible:

- suspender;
- reanudar.

8.1.4 Guión de lanzamiento

8.1.4.1 Visión general

- Información de gestión que representa una serie de instrucciones en lenguajes de escritura de guiones especializados.
- Cada guión debe ser independiente.

8.1.4.2 Características del guión de lanzamiento

La clase guión de lanzamiento tiene los siguientes atributos.

8.1.4.2.1 Identificador de guión

El valor de este atributo identifica un caso de la clase de objeto gestionado guión de lanzamiento.

8.1.4.2.2 Tipo de resultado de ejecución

El valor de este atributo identifica el tipo esperado de resultado de ejecución.

8.1.4.2.3 Estado administrativo

Se define en la Rec. X.731 del CCITT | ISO/CEI 10164-2.

8.1.4.3 Lotes de la clase de objeto guión de lanzamiento

La clase de objeto gestionado guión de lanzamiento tiene el siguiente lote obligatorio:

- lote de guión de lanzamiento (launch script package).

8.1.5 Clase generador básico

8.1.5.1 Visión general

- Significa la capacidad de creación de nuevos casos de objetos contenidos con generación de nombre automática para estos nuevos casos.
- Superclase de las clases de objeto hilo y lanzador del secuenciador de instrucciones.

8.1.5.2 Lotes de la clase generador básico

La clase generador básico tiene el siguiente lote obligatorio:

- lote de generador básico (basic spawner package).

8.1.6 Lanzador

8.1.6.1 Visión general

- Subclase de las clases de objetos gestionados generador básico y referenciador de guiones.
- Iniciador de la ejecución del guión de lanzamiento al recibir una activación.
- Actúa como IVMO para un hilo.
- Un guión es ejecutado por el lanzador por medio de uno o más hilos.

8.1.6.2 Lotes de la clase de objeto gestionado lanzador

Los lotes obligatorios de la clase de objetos gestionados lanzador son:

- lote de lanzador (launch pad package);
- aceptador de acciones de activador (trigger action accepter);
- pasador de parámetros (parameter passer);
- lote de resultado de activación (trigger result package);
- aceptador de eventos de activación (trigger event accepter);
- aceptador de terminación (terminate accepter);
- planificador externo (external scheduler);
- aceptador de suspensión y reanudación (suspend resume accepter).

8.1.6.3 Características del lanzador

La clase de objeto gestionado lanzador tiene los siguientes atributos.

8.1.6.3.1 Lista de guiones disponibles

Ésta es una lista de todos los guiones que un lanzador es capaz de ejecutar. El lanzador ejecuta solamente aquellos guiones que están presentes en el componente lista de parámetros de ejecución de los parámetros del activador y en el atributo lista de guiones disponibles.

8.1.6.3.2 Lista de parámetros de ejecución por defecto

Ésta es una lista de identificadores de guiones y parámetros de guiones que se utilizan para la ejecución por defecto cuando un lanzador es activado sin parámetros.

8.1.6.3.3 Estado administrativo

Este atributo representa la capacidad administrativa del lanzador de ejecutar su función. Se definen los siguientes estados administrativos:

a) Desbloqueado – El lanzador está autorizado a comenzar o reanudar la ejecución de guiones de lanzamiento.

b) Bloqueado – El lanzador no está autorizado a comenzar la ejecución de guiones de lanzamiento. Si hay ejecuciones en curso, éstas se supenden.

8.1.6.3.4 Estado operacional

Este atributo representa la capacidad operacional del lanzador de ejecutar sus funciones.

Se definen los siguientes estados operacionales:

a) Habilitado – El lanzador es operacional y está disponible para ejecutar un guión.

b) Inhabilitado – El lanzador no es operacional y no está disponible para ejecutar guiones.

8.1.6.3.5 Estado de utilización

Este atributo representa el estado de utilización del lanzador. Se definen los siguientes estados de utilización:

a) Ocupado – El lanzador está siendo usado para ejecutar un guión de lanzamiento.

b) Reposo – El lanzador no está siendo usado para ejecutar un guión.

8.1.6.3.6 Estados de disponibilidad

Esta condición de estado indica si el lanzador está disponible para ejecutar su función. Se definen los siguientes estados:

a) Fuera de servicio

El lanzador ha sido puesto en estado inactivo por un proceso de control interno de acuerdo con una planificación horaria predeterminada.

b) No está fuera de servicio

El atributo estado de disponibilidad no tiene el valor “fuera de servicio”, por lo que está activo.

8.1.6.3.7 Caso de objeto observado

Se define en la Rec. UIT-T X.739 | ISO/CEI 10164-11.

8.1.6.3.8 Identificador de atributo observado

Se define en la Rec. UIT-T X.739 | ISO/CEI 10164-11.

8.1.6.3.9 Estado de control

Este atributo se define en Rec. X.731 del CCITT | ISO/CEI 10164-2. El valor por defecto es vacío. Si se suspende la ejecución, el estado de control cambia a "suspendido" y si se reanuda la ejecución, el valor cambia a vacío.

8.1.6.3.10 Identificador de lanzador

Este atributo denomina un caso de la clase de objeto gestionado lanzador.

8.1.6.4 Notificaciones de la clase de objeto gestionado lanzador

La clase de objeto gestionado lanzador tiene las siguientes notificaciones, que pueden ser enviadas al destino o destinos de notificación apropiados:

- resultado de activación, según se define en 8.2.1;
- alarma de error de procesamientos según se define en la Rec. X.733 del CCITT | ISO/CEI 10164-4.

8.1.7 Lanzador asíncrono

8.1.7.1 Visión general

- Subclase de la clase de objeto gestionado lanzador.
- Los hilos suspendibles son generados por el lanzador asíncrono.
- Envía una notificación de resultado de activación tan pronto como los hilos suspendibles son generados.

8.1.7.2 Lotes de la clase de objeto gestionado lanzador asíncrono

8.1.7.3 Clase de objeto gestionado

La clase de objeto gestionado lanzador asíncrono tiene el siguiente lote:

- lote de resultado asíncrono de activación (triggerAsynchronousResultPackage).

8.1.8 Lanzador síncrono

8.1.8.1 Visión general

- Subclase de la clase de objeto gestionado lanzador.
- Los hilos son generados por el lanzador síncrono.
- Envía una notificación de resultado de activación después de sincronizar los resultados tan pronto como todos los hilos están completos.

8.1.8.2 Lotes de la clase de objeto gestionado lanzador síncrono

La clase de objeto gestionado lanzador síncrono tiene el siguiente lote:

- lote de resultado síncrono de activación (trigger synchronous result package).

8.1.9 Guión de cadena general

8.1.9.1 Visión general

- Subclase de la clase de objeto guión de lanzamiento.
- Los guiones pueden ser representados en forma de una cadena general.
- Es posible añadir otros guiones especializados como subclases.

8.1.9.2 Características del lenguaje de escritura de guiones de cadena general

La clase de objeto gestionado lenguaje de escritura de guiones de cadena general tiene los siguientes atributos.

8.1.9.2.1 Nombre del lenguaje de escritura de guión

Éste es el nombre del lenguaje que define las propiedades sintácticas y semánticas de un guión que se representa como una cadena general.

8.1.9.2.2 Contenido de guión

Indica el guión que se representa como una cadena general.

8.1.9.3 Lotes del lenguaje de escritura de guión de cadena general

La clase de objeto gestionado lenguaje de escritura de guión de cadena general tiene el siguiente lote obligatorio:

- lote de guión de cadena general (general string script package).

8.1.10 Referenciador de guión

8.1.10.1 Visión general

- Superclase para las clases de objetos gestionados lanzador y hilo.
- Define una correspondencia de relación de referencia entre lanzador y guión de lanzamiento e hilo y guión de lanzamiento.

8.1.10.2 Lotes del referenciador de guión

El referenciador de guión tiene el siguiente lote obligatorio:

- lote de referenciador de guión (script referencer package).

8.2 Notificaciones genéricas

En esta Especificación se definen las siguientes notificaciones.

8.2.1 Resultado de activación

Devuelve el resultado de una ejecución de un guión y con el valor código de error puesto a "ningún error" si la ejecución tuvo éxito o un código de error apropiado para indicar la naturaleza del fallo. Los códigos de error que se pueden devolver en el campo código de error de la notificación resultado de ejecución son:

- ningún error, si la ejecución tuvo éxito;
- ningún error de guión, si la ejecución falló porque no se especificó el nombre del guión;
- error guión rechazado, si un guión no estaba en la lista de guiones que un lanzador está configurado para ejecutar;
- error tipo de parámetro no válido, si el tipo de parámetro esperado por el guión y el suministrado por el guión no concuerdan;
- error valor de parámetro no válido, si el valor suministrado en el parámetro no es válido, por ejemplo, fuera de gama;
- error de sintaxis de guión, si la ejecución del guión fracasó debido a un error de sintaxis en el guión;
- error ejecución de guión fracasada, si la ejecución fracasó por razones distintas a una sintaxis inadecuada;
- número de parámetros no válidos, si el número de parámetros suministrados no concuerda con el número de parámetros esperados por el guión;
- error de acceso no autorizado, si se ha negado el acceso a uno o más objetos que el guión había de utilizar.

8.2.2 Resultado de ejecución

Devuelve el resultado de la ejecución de un guión y con el valor código de error puesto a “ningún error” si tuvo éxito o un código de error apropiado para indicar la naturaleza del fallo. Los códigos de error que se pueden devolver en el campo código de error de la notificación resultado de ejecución son:

- ningún error, si la ejecución tuvo éxito;
- ningún error de guión, si la ejecución falló porque no se especificó el nombre del guión;
- error guión rechazado, si un guión no estaba en la lista de guiones que un lanzador está configurado para ejecutar;
- error tipo de parámetro no válido, si el tipo de parámetro esperado por el guión y el suministrado por el guión no concuerdan;
- error valor de parámetro no válido, si el valor suministrado en el parámetro no es válido, por ejemplo, fuera de gama;
- error de sintaxis de guión, si la ejecución del guión fracasó debido a un error de sintaxis en el guión;
- error ejecución de guión fracasada, si la ejecución fracasó por razones distintas a una sintaxis inadecuada;
- número de parámetros no válido, si el número de parámetros suministrados no concuerda con el número de parámetros esperados por el guión;
- error de acceso no autorizado, si se ha negado el acceso a uno o más objetos que el guión había de utilizar.

8.3 Acciones genéricas

Los siguientes tipos de acciones se definen en esta Especificación. Estas acciones se han definido para los objetos gestionados lanzador y hilo suspendible.

8.3.1 Acción suspender

La acción suspender dirigida a un objeto gestionado hace que el objeto suspenda la ejecución del guión que está ejecutando en ese momento. Los parámetros transportados por la acción suspender son el identificador de activador para identificar esta acción y el identificador de hilo si la acción está dirigida a un hilo o el identificador de lanzador, si esta acción está dirigida a un lanzador.

8.3.2 Acción reanudar

La acción reanudar dirigida a un objeto gestionado hace que el objeto suspenda la ejecución del guión que está ejecutando en ese momento. Los parámetros transportados por la acción suspender son el identificador de activador para identificar esta acción y el identificador de hilo si la acción está dirigida a un hilo o el identificador de lanzador, si esta acción está dirigida a un lanzador.

8.3.3 Acción terminar

La acción terminar dirigida a un objeto gestionado origina la terminación incondicional de cualquier ejecución de guión por ese objeto y cualesquiera objetos generados por éste. El parámetro identificador de activador se utiliza para identificar esta acción.

8.3.4 Acción activar

Una acción activar inicia la ejecución de un guión. Suministra el parámetro identificador de activador que identifica esta acción y su componente lista de parámetros de ejecución consiste en una secuencia de identificadores de guión que indica los guiones que deben ser ejecutados y los parámetros de guión que se necesitan para ejecutar estos guiones.

9 Servicios

9.1 Introducción

El secuenciador de instrucciones proporciona servicios para modificar las operaciones de secuenciador de instrucciones y de guiones de lanzamiento. En particular, las operaciones que se pueden aplicar a cada caso de un secuenciador de instrucciones y guiones de lanzamiento son:

- creación de casos de secuenciador de instrucciones, lanzador y de guión de lanzamiento;
- supresión de casos de lanzador y guión de lanzamiento;
- modificación de atributos de secuenciador de instrucciones, lanzador y guión de lanzamiento;
- extracción de atributos de secuenciador de instrucciones, lanzador y guión de lanzamiento.

Además de los servicios de modificación mencionados, esta función proporciona servicios de notificación y acción para activar la ejecución de instrucciones.

9.2 Servicios de iniciación, terminación, modificación y extracción

Los servicios PT-CREACIÓN, PT-SUPRESIÓN, PT-FIJACIÓN y PT-OBTENCIÓN se pueden utilizar para crear, suprimir, modificar y extraer valores y atributos del objeto soporte de gestión de secuenciador de instrucciones y de los objetos gestionados lanzador y guión de lanzamiento.

9.3 Servicios de notificación

9.3.1 Definición del servicio resultado de ejecución

Esta subcláusula especifica el servicio informe de resultado de ejecución que se define en la presente Recomendación | Norma Internacional y su correspondencia con el servicio M-INFORME DE EVENTO de CMIS.

Cuadro 6 – Parámetros de informe de resultado de ejecución

Nombre de parámetro	pet./ind.	resp./conf.
Invoke identifier	P	P
Mode	P	–
Managed object class	P	P
Managed object instance	P	P
Event type	M	C (=)
Event time	P	–
Event information		
trigger id	M	–
script id	M	–
thread id	M	–
error code	U	–
execution result type	U	–
execution result	U	–
Current time	–	P
Event reply	–	P
Errors	–	P

Cuadro 7 – Parámetros de informe de resultado de activación

Nombre de parámetro	pet./ind.	resp./conf.
Invoke identifier	P	P
Mode	P	–
Managed object class	P	P
Managed object instance	P	P
Event type	M	C (=)
Event time	P	–
Event information		
trigger id	M	–
script id	U	–
thread id	–	–
script parameters	U	–
execution type	U	–
error code	U	–
execution result	U	–
Current time	–	P
Event reply	–	P
Errors	–	P

9.4 Servicios de acción

Esta subcláusula especifica los servicios de acción activar y terminar que se definen en la presente Recomendación | Norma Internacional y su correspondencia con el servicio CMIS M-EVENTO DE ACCIÓN.

Cuadro 8 – Parámetros del servicio de acción activar

Nombre de parámetro	pet./ind.	resp./conf.
Invoke identifier	P	P
Linked identifier	–	P
Mode	P	–
Base object class	P	–
Base object instance	P	–
Scope	P	–
Filter	P	–
Managed object class	–	P
Managed object instance	–	P
Access control	P	–
Synchronization	P	–
Action type	M	–
Action Information		
trigger id	M	–
Attribute list	U	–
Errors	–	P

Cuadro 9 – Parámetros del servicio de acción terminar

Nombre de parámetro	pet./ind.	resp./conf.
Invoke identifier	P	P
Linked identifier	–	P
Mode	P	–
Base object class	P	–
Base object instance	P	–
Scope	P	–
Filter	P	–
Managed object class	–	P
Managed object instance	–	P
Access control	P	–
Synchronization	P	–
Action type	M	–
Action Information		
trigger id	M	–
Errors	–	P

Cuadro 10 – Parámetros del servicio de acción suspender

Nombre de parámetro	pet./ind.	resp./conf.
Invoke identifier	P	P
Linked identifier	–	P
Mode	P	–
Base object class	P	–
Base object instance	P	–
Scope	P	–
Filter	P	–
Managed object class	–	P
Managed object instance	–	P
Access control	P	–
Synchronization	P	–
Action type	M	–
Action Information		
trigger id	M	–
thread id	U	–
launch pad id	U	–
Errors	–	P

Cuadro 11 – Parámetros del servicio de acción reanudar

Nombre de parámetro	pet./ind.	resp./conf.
Invoke identifier	P	P
Linked identifier	–	P
Mode	P	–
Base object class	P	–
Base object instance	P	–
Scope	P	–
Filter	P	–
Managed object class	–	P
Managed object instance	–	P
Access control	P	–
Synchronization	P	–
Action type	M	–
Action Information		
trigger id	M	–
thread id	U	–
launch pad id	U	–
Errors	–	P

10 Unidades funcionales

En la presente Recomendación | Norma Internacional de definen tres unidades funcionales para la gestión de secuenciadores de instrucciones:

a) *Unidad funcional de ejecución*

La unidad funcional de ejecución requiere los servicios de PT-CREACIÓN, el servicio de notificación de resultado de ejecución y el servicio de informe de alarmas de error de procesamiento de la acción activar.

b) *Unidad funcional de supervisión*

La unidad funcional de supervisión requiere los servicios de PT-OBTENCIÓN.

c) *Unidad funcional de control*

La unidad funcional de control requiere los servicios de la acción terminar, PT-SUPRESIÓN.

11 Protocolos y sintaxis abstracta

11.1 Sintaxis abstracta

11.1.1 Objetos gestionados

11.1.1.1 Objetos gestionados definidos

El cuadro 12 muestra la relación entre los objetos gestionados definidos en 8.1 y la especificación de clase de objetos gestionados en el anexo A.

Cuadro 12 – Objetos gestionados y etiquetas de referencia

Nombre de objeto gestionado	Etiqueta de referencia
Basic spawner	basicSpawnerClass
Suspendable thread	suspendableThread
Thread	thread
Asynchronous launch pad	asynchronousLaunchPad
Synchronous launch pad	synchronousLaunchPad
Launch pad	launchPad
Launch script	launchScript
Command sequencer	commandSequencer
General string script	generalStringScript
Script Referencer	scriptReferencer

11.2 Atributos

11.2.1 Atributos importados de la definición de información de gestión

La presente Recomendación hace referencia a los siguientes atributos de gestión, cuyas sintaxis abstractas se especifican en la Rec. X.721 del CCITT | ISO/CEI 10165-2:

a) estados administrativo (administrativeState);

b) estado de utilización (usageState);

c) estado operacional (operationalState);

d) estado de control (controlStatus);

e) estado de disponibilidad (availabilityStatus).

Hace referencia también a los siguientes atributos de gestión, cuyas sintaxis abstractas se especifican en la Rec. UIT-T X.739 | ISO/CEI 10164-11:

a) caso de objeto observado (observedObjectInstance);

b) identificador de atributo observado (observedAttributeId).

11.2.2 Atributos definidos en esta Especificación

La presente Recomendación define los siguientes atributos de gestión cuyas sintaxis abstractas se especifican en el anexo A:

a) identificador de lanzador (launchPadId);

b) identificador de secuenciador de instrucciones (commandSequencerId);

c) identificador de hilo (threadId);

d) identificador de guión (scriptId);

e) identificador de activador (triggerId);

f) tipo de resultado de ejecución (executionResultType);

g) parámetros de ejecución (executingParameters);

h) nombre de lenguaje de guión (scriptLanguageName);

i) contenido de guión (scriptContent);

j) lista de parámetros de ejecución por defecto (defaultExecutionParameterList);

k) lista de guiones disponibles (availableScriptList).

11.2.3 Correspondencia de parámetros con atributos

El cuadro 13 muestra la relación entre los parámetros de servicio definidos en 8.1 y 8.2 y las especificaciones de tipos de atributos del anexo A.

Cuadro 13 – Parámetros y nombres de atributo

Parámetro	Nombre de atributo
Launch pad id	launchPadId
Command sequencer id	commandSequencerId
Thread id	threadId
Script id	scriptId
Trigger id	triggerId
Execution result type	executionResultType
Executing parameters	executingParameters
Script language name	scriptLanguageName
Script content	scriptContent
Default execution parameter list	defaultExecutionParameterList
Available script list	availableScriptList

11.4 Notificaciones

11.4.1 Notificaciones referenciadas

La presente Recomendación hace referencia a los siguientes eventos definidos en la Rec. X.730 del CCITT | ISO/CEI 10164-1:

a) notificación de creación de objeto (object creation notification);

b) notificación de supresión de objeto (object deletion notification);

c) notificación de alarma de error de procesamiento (processing error alarm notification).

Esta Especificación hace referencia también al siguiente evento definido en la Rec. X.731 del CCITT | ISO/CEI 10164-2:

– notificación de cambio de estado (state change notification).

11.4.2 Notificaciones definidas en esta Especificación

El cuadro 14 muestra la relación entre las notificaciones en 9.3 y las especificaciones de tipos de notificación del anexo A.

Cuadro 14 – Notificaciones

Tipo de evento	Tipo de notificación
Trigger result	triggerResultInfo
Execution result	executionResultInfo

11.5 Acciones

11.5.1 Acciones definidas en esta Especificación

El cuadro 15 muestra la relación entre las acciones definidas en 9.4 y las especificaciones de tipos de notificación del anexo A.

Cuadro 15 – Acciones

Nombre de acción	Etiqueta de referencia
terminate	terminate
suspend	suspend
resume	resume
trigger	trigger

11.6 Negociación de unidades funcionales

La presente Recomendación | Norma Internacional asigna el siguiente valor de identificador de objeto:

{joint-iso-itu-t ms(9) function(2) part21(21) functionalUnitPackage(1)}

como un valor del FunctionalUnitPackageId del tipo ASN.1 de la Rec. UIT-T X.701 del CCITT | ISO/CEI 10040 que se ha de utilizar para negociar las siguientes unidades funcionales:

0 unidad funcional de ejecución (Execution functional unit)

M unidad funcional de supervisión (Monitoring functional unit)

2 unidad funcional de control (Control functional unit)

donde el número identifica la posición de bits en la CADENA DE BITS asignada a las unidades funcionales, y los nombres que hacen referencia a las unidades funcionales se definen en la cláusula 10.

En el contexto de aplicación de gestión de sistemas, el mecanismo para negociar las unidades funcionales se describe en la Rec. X.701 del CCITT | ISO/CEI 10040.

NOTA – El requisito para negociar unidades funcionales es especificado por el contexto de aplicación.

12 Relación con otras funciones

El secuenciador de instrucciones utiliza los servicios definidos en la Rec. X.731 del CCITT | ISO/CEI 10164-2 para la notificación de cambios de estado, los servicios definidos en la Rec. X.730 del CCITT | ISO/CEI 10164-1 para la creación y supresión de objetos gestionados, la extracción de atributos y la notificación de cambios de valores de atributo.

El secuenciador de instrucciones utiliza los servicios definidos en la Rec. UIT-T X.741 | ISO/CEI 10164-9 para proporcionar capacidades de control de acceso a objetos gestionados que pueden ser accionados por hilos.

13 Conformidad

Hay dos clases de conformidad: clase de conformidad general y clase de conformidad dependiente. Un sistema que alega realizar los elementos de procedimiento para servicios de gestión de sistemas referenciados en esta Especificación cumplirá los requisitos de la clase de conformidad general o de la clase de conformidad dependiente definidos en las siguientes subcláusulas. El suministrador de la realización indicará la clase con la cual se alega conformidad.

13.1 Requisitos de clase de conformidad general

Un sistema que alega conformidad general admitirá esta función para todas las clases de objetos gestionados que importan la información de gestión definida en esta Especificación.

NOTA – Esto es aplicable a todas las subclases de clases de objetos de soporte de gestión definidos en esta Especificación.

13.1.1 Conformidad estática

El sistema:

a) admitirá el cometido de gestor o de agente, o ambos, con respecto a la unidad funcional de medida de control y a la unidad funcional de medida de supervisión;

b) admitirá la sintaxis de transferencia derivada de las reglas de codificación especificadas en la Rec. X.209 del CCITT | ISO/CEI 8825 y denominada {joint-iso-itu-t asn1(1) basicEncoding(1)}, para generar e interpretar las unidades de datos de protocolo de aplicación de gestión, definidas por los tipos de datos abstractos referenciados en 11.4 y 11.5;

c) cuando actúa con el cometido de agente, admitirá uno o más casos por lo menos de las clases de objetos gestionados secuenciador de instrucciones, lanzador, guión de lanzamiento, hilo, o cualquiera de sus subclases.

13.1.2 Conformidad dinámica

En los cometidos para los cuales alega conformidad, el sistema:

a) Sustentará los elementos de procedimiento definidos en:

- la Rec. X.730 del CCITT | ISO/CEI 10164-1 para los servicios PT-OBTENCIÓN, PT-CREACIÓN, PT-SUPRESIÓN, PT-FIJACIÓN, informe de creación de objeto, informe de supresión de objeto e informe de cambio de atributos;
- la Rec. X.731 del CCITT | ISO/CEI 10164-2 para el servicio de informe de cambio de estado;
- la Rec. X.733 del CCITT | ISO/CEI 10164-4 para el servicio de informe de alarma de error de procedimiento.

b) Sustentará los elementos de procedimiento definidos en la presente Especificación para los siguientes servicios de informe y acción:

- notificación de resultado de ejecución (execution result notification);
- acción activar (trigger action);
- acción terminar (terminate action);
- acción suspender (suspend action);
- acción reanudar (resume action).

13.2 Requisitos de clase de conformidad dependiente

13.2.1 Conformidad estática

El sistema:

a) admitirá la sintaxis de transferencia derivada de las reglas de codificación especificadas en la Rec. X.209 del CCITT | Norma ISO/CEI 8825 y denominada {joint-iso-itu-t asn1(1) basicEncoding(1)}, para generar e interpretar las unidades de datos de protocolo de aplicación de gestión, definidas por los tipos de datos abstractos referenciados en la cláusula 11.1, requeridos por una especificación de referencia;

b) admitirá uno o más casos de una de las clases de objetos gestionados secuenciador de instrucciones, lanzador, guión de lanzamiento, hilo o cualquiera de sus subclases, cuando actúan con el cometido de agente.

13.2.2 Conformidad dinámica

El sistema sustentará los elementos de procedimiento referenciados por la presente Especificación, requeridos por una especificación de referencia.

13.3 Conformidad para sustentar definiciones de objetos gestionados

Los objetos secuenciador de instrucciones sustentados por el sistema abierto satisfarán el comportamiento especificado en la cláusula 8 y la sintaxis especificada en el anexo A.

Annex A

Definition of Management Information

(This annex forms an integral part of this Recommendation | International Standard)

A.1 Managed object class definitions

A.1.1 Basic objects

```
basicSpawnerClass   MANAGED OBJECT CLASS
DERIVED FROM "CCITT Rec. X.721 | ISO/IEC 10165-2:1992":top;
CHARACTERIZED BY basicSpawnerPackage ;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx1(1)};

commandSequencer    MANAGED OBJECT CLASS
DERIVED FROM "CCITT Rec. X.721 | ISO/IEC 10165-2:1992":top;
CHARACTERIZED BY
       commandSequencerPackage   PACKAGE
       BEHAVIOUR commandSequencerBehaviour BEHAVIOUR
       DEFINED AS "An instance of this class represents a resource acting in a manager
       role as an invoker of operations determined by its launch scripts.";;
       ATTRIBUTES
              commandSequencerId    GET,
       "CCITT Rec. X.731|ISO/IEC 10164-2:1992": administrativeState GET-REPLACE,
              "CCITT Rec. X.731|ISO/IEC 10164-2:1992":operationalState GET;
       NOTIFICATIONS
              "CCITT Rec. X.730 | ISO/IEC 10164-1": objectCreation,
              "CCITT Rec. X.730 | ISO/IEC 10164-1": objectDeletion,
              "CCITT Rec. X.731 | ISO/IEC 10164-2": stateChange;;;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21)
managedObjectClass(3) xx2(2)};

generalStringScript MANAGED OBJECT CLASS
DERIVED FROM launchScript;
CHARACTERIZED BY generalStringScriptPackage;;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx3(3)};

asynchronousLaunchPad MANAGED OBJECT CLASS
DERIVED FROM launchPad;
CHARACTERIZED BY triggerAsynchronousResultPackage;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx4(4)};

synchronousLaunchPad MANAGED OBJECT CLASS
DERIVED FROM launchPad;
CHARACTERIZED BY triggerSynchronousResultPackage;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx5(5)};
```

```
launchPad    MANAGED OBJECT CLASS
DERIVED FROM basicSpawnerClass, scriptReferencer;
CHARACTERIZED BY
           launchPadPackage,
             triggerActionAccepter,
             parameterPasser,
             triggerResultPackage,
             triggerEventAccepter,
             terminateAccepter,
             "CCITT Rec. 721 | ISO/IEC 10165-2:1992": externalScheduler,
              suspendResumeAccepter;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx6(6)};

launchScript MANAGED OBJECT CLASS
DERIVED FROM "CCITT Rec. X.721 | ISO/IEC 10165-2:1992":top;
CHARACTERIZED BY
       launchScriptPackage PACKAGE
       BEHAVIOUR launchScriptBehaviour BEHAVIOUR
       DEFINED AS "This managed object represents instructions to be carried out by a
       command sequencer.";;
       ATTRIBUTES
              scriptId GET,
              executionResultType GET,
              "CCITT Rec. X.721 | ISO /IEC 10165-2:1992": administrativeState GET-
              REPLACE;;;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx7(7)};

--
-- The following non-instantiable superclass simplifies the description of the
-- relationship between a launch pad and its scripts, along with the description
-- of the relationship between threads and scripts. Both the launch pad and
-- thread classes include it in their inheritance hierarchies.
--
scriptReferencer MANAGED OBJECT CLASS
DERIVED FROM "ITU-T Rec. X.725 | ISO/IEC 10165-7": genericRelationshipObject;
CHARACTERIZED BY scriptReferencerPackage;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx8(8)};

thread MANAGED OBJECT CLASS
       DERIVED FROM basicSpawnerClass, scriptReferencer;
       CHARACTERIZED BY threadPackage, executionResultPackage;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx9(9)};

suspendableThread MANAGED OBJECT CLASS
       DERIVED FROM thread;
       CHARACTERIZED BY suspendResumeAccepter;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3)
xx10(10)};
```

A.2 Package definitions

A.2.1 Basic packages

```
basicSpawnerPackage  PACKAGE
        BEHAVIOUR spawnerBehaviour;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx1(1)};

generalStringScriptPackage PACKAGE
        BEHAVIOUR generalStringScriptBehaviour;
        ATTRIBUTES scriptLanguageName GET-REPLACE,
                scriptContent GET-REPLACE;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx2(2)};

parameterPasser PACKAGE
        BEHAVIOUR
        parameterPasserBehaviour;
        ATTRIBUTES
            "CCITT Rec. X.721 | ISO /IEC 10165-2:1992": administrativeState,
            "CCITT Rec. X.721 | ISO /IEC 10165-2:1992": operationalState,
            "CCITT Rec. X.721 | ISO /IEC 10165-2:1992": usageState,
            "CCITT Rec. X.721 | ISO /IEC 10165-2:1992": availabilityStatus;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx3(3)};

executionResultPackage   PACKAGE
        BEHAVIOUR executionResultBehaviour;;
        NOTIFICATION executionResultInfo;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx4(4)};

launchPadPackage PACKAGE
        BEHAVIOUR launchPadBehaviour;
        ATTRIBUTES launchPadId GET;
        NOTIFICATIONS
        "CCITT Rec. X.721 | ISO/IEC 10165-2:1992": processingErrorAlarm;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx5(5)};

scriptReferencerPackage PACKAGE
        BEHAVIOUR scriptReferencerBehaviour;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx6(6)};

suspendResumeAccepter PACKAGE
        BEHAVIOUR suspendResumeBehaviour;
        ATTRIBUTES "CCITT Rec. X.721 | ISO/IEC 10165-2:1992":controlStatus GET;
        ACTIONS suspend, resume;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx7(7)};
```

```
terminateAccepter PACKAGE
       ACTIONS terminate;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx8(8)};

threadPackage PACKAGE
        BEHAVIOUR threadBehaviour,
                   simpleScriptExecutionBehaviour;
       ATTRIBUTES scriptId GET,
                  threadId GET,
                  executingParameters   GET SET-BY-CREATE,
                       "CCITT Rec. X.731|ISO/IEC 10164-2:1992": operationalState GET;
        NOTIFICATIONS
                "CCITT Rec. X.734|ISO/IEC 10164-5": processingErrorAlarm;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx9(9)};

triggerActionAccepter PACKAGE
        BEHAVIOUR   spawnerBehaviour,
                      triggerActionAccepterBehaviour;;
        ATTRIBUTES defaultExecutionParameterList REPLACE WITH DEFAULT
                   GET-REPLACE
                   SET BY CREATE
                   DEFAULT VALUE CSModule.emptyExecutionParameterList;
                availableScriptList REPLACE WITH DEFAULT
                   ADD-REMOVE
                   GET-REPLACE
                   SET BY CREATE
                   DEFAULT VALUE CSModule.emptyScriptList;
        ACTIONS Trigger;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx10(10)};

triggerEventAccepter PACKAGE
        BEHAVIOUR triggerEventAccepterBehaviour;
        ATTRIBUTES
           "ITU-T Rec. X.739 (1993)|ISO/IEC 10164-11:1994": observedObjectInstance GET-
REPLACE,
           "ITU-T Rec. X.739 (1993)|ISO/IEC 10164-11:1994": observedAttributeId GET-
REPLACE;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx11(11)};

triggerAsynchronousResultPackage PACKAGE
        BEHAVIOUR triggerAsynchronousResultBehaviour;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx12(12)};

triggerSynchronousResultPackage PACKAGE
        BEHAVIOUR triggerSynchronousResultBehaviour;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx13(13)};
```

```
triggerResultPackage PACKAGE

        BEHAVIOUR    triggerResultBehaviour;;

        NOTIFICATION triggerResultInfo;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx14(14)};
```

A.3 Behaviour definitions

```
spawnerBehaviour  BEHAVIOUR

       DEFINED AS !Instances of this class are capable of causing the creation of new
       object instances. The newly created instances will be contained by this instance,
       and their names will be automatically generated. Until all created objects are
       complete, this object's usage status will be "in use". If this object's
       administrative state is "locked" such new objects cannot be created. If, due to
       local resource limitations, this object is incapable of supporting more contained
       objects, its usage status will be "busy".


       When an instance of this class causes the creation of new objects, it serves as an
       IVMO during the creation by supplying values for the new object's attributes based
       on its own defaultExecutionParameterList attribute or any parameters which were
       supplied to it as part of the action or local mechanism which triggered the
       spawning of the new instance.


       An instance of this class may cause the creation of new object instances from a
       single script id, from a set of script ids in any order or from a sequence of
       script ids in the order specified in the list, i.e. after the first has been
       created the second may not be created until after the first is completed, and so
       on. The value of the created object's scriptId attribute gets its value from the
       corresponding element of this object's script list.!;


executionResultBehaviour BEHAVIOUR

DEFINED AS "Instances of a class supporting this behaviour report intermediate and final
results from execution of a thread.";


triggerAsynchronousResultBehaviour BEHAVIOUR

       DEFINED AS "As soon as all threads that must be launched by one trigger is
       launched, the launch pad issues the triggerResultInfo notification.";


triggerSynchronousResultBehaviour BEHAVIOUR

       DEFINED AS "As soon as all threads that must be launched by one trigger have
       completed, the launch pad issues the triggerResultInfo notification which contains
       execution results or errors.";


triggerResultBehaviour BEHAVIOUR

       DEFINED AS "The launch pad issues the triggerResultInfo notification. ";


scriptReferencerBehaviour BEHAVIOUR

       DEFINED AS "A script referencer is a non-instantiable object class which defines a
       reference relationship mapping from instances of the launch pad and the launch
       script managed object classes and from instances of the thread and the launch
       script managed object classes.";


suspendResumeBehaviour BEHAVIOUR

       DEFINED AS "Execution of a script by a thread may be suspended by a suspend action
       directed at the thread or launch pad and subsequently resumed by a resume action.
       Default value of controlStatus is empty. If the suspend action is performed, the
       value changes to suspended. After the resume action is performed, the value changes
       back to empty.";
```

```
triggerEventAccepterBehaviour BEHAVIOUR

      DEFINED AS "The launch pad has attributes to monitor a specific attribute in a
      specific object instance. If the value of the monitored attribute is changed, a
      trigger to launch the scripts specified by the script ids specified by the default
      execution parameter list attribute is generated. In the case that the monitored
      attribute is a counter of EDC (Event Discrimination Counter) defined in Annex C,
      the notifications through the EDC trigger the launching of script execution by the
      launch pad.";


triggerActionAccepterBehaviour BEHAVIOUR

      DEFINED AS "When an instance of this class which is on duty receives a trigger,
      from a trigger activator, if its scriptId attribute is not empty, a new object is
      created in which the script id and class of the new instance come from the value of
      this instance's scriptId attribute and any of its other attributes and any
      parameters carried by the trigger.";


threadBehaviour BEHAVIOUR

      DEFINED AS "When an instance of an object of this class is created, it begins
      execution of the command sequence specified through its attributes, using its
      parameter list to supply any parameters needed by the script. When execution of
      this sequence is complete, the object is deleted. If execution of the script causes
      the creation of contained threads, this thread is not considered complete until all
      contained threads are complete.";


parameterPasserBehaviour BEHAVIOUR

      DEFINED AS "An instance of an object of this class passes a set of parameters to an
      instance of an object of another class.";


simpleScriptExecutionBehaviour BEHAVIOUR
      DEFINED AS "A script is executed or interpreted by local means. Its execution
      status mirrors the following states:

            – activated (spontaneous transition to next state: executing);

            – executing (next state: timed out or completed);

            – timed out (spontaneous transition to next state: completed);

            – completed.

       NOTE – Timeout value is implementation dependent.";


generalStringScriptBehaviour BEHAVIOUR

      DEFINED AS "The syntax and semantics of scripting language which can be represented
      as general string. See Annexes F and G for details on one such language, SMSL.";
```

A.4 Attribute definitions

```
availableScriptList  ATTRIBUTE

       WITH ATTRIBUTE SYNTAX CSModule.AvailableScriptList;

       MATCHES FOR EQUALITY;

       BEHAVIOUR

       availableScriptListBehaviour BEHAVIOUR

       DEFINED AS "A set of managed object instance names of the script instructions
       which can be executed by a launch pad.";;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx1(1)};


commandSequencerId  ATTRIBUTE

      WITH ATTRIBUTE SYNTAX CSModule.CommandSequencerId;

      MATCHES FOR EQUALITY;

      BEHAVIOUR
```

```
       commandSequencerIdBehaviour   BEHAVIOUR
       DEFINED AS "The managed object instance name of the command sequencer.";;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx2(2)};

executionResultType ATTRIBUTE
       WITH ATTRIBUTE SYNTAX CSModule.ExecutionResultType;
       MATCHES FOR EQUALITY;
       BEHAVIOUR
       executionResultTypeBehaviour BEHAVIOUR
       DEFINED AS "This indicates the type of execution result.";;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx3(3)};

scriptContent ATTRIBUTE
       WITH ATTRIBUTE SYNTAX CSModule.ScriptContent;
       MATCHES FOR EQUALITY;
       BEHAVIOUR
       scriptContentBehaviour BEHAVIOUR
       DEFINED AS "The contents of a launch script represented by a general string.";;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx4(4)};

scriptId   ATTRIBUTE
       WITH ATTRIBUTE SYNTAX CSModule.ScriptId;
       MATCHES FOR EQUALITY;
       BEHAVIOUR
       scriptIdBehaviour   BEHAVIOUR
       DEFINED AS "The managed object instance name of the script to be executed.";;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx5(5)};

launchPadId  ATTRIBUTE
       WITH ATTRIBUTE SYNTAX CSModule.LaunchPadId;
       MATCHES FOR EQUALITY;
       BEHAVIOUR
       launchPadIdBehaviour   BEHAVIOUR
       DEFINED AS "The managed object instance name of the launch pad.";;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx6(6)};

scriptLanguageName ATTRIBUTE
       WITH ATTRIBUTE SYNTAX CSModule.ScriptLanguageName;
       MATCHES FOR EQUALITY;
       BEHAVIOUR
       scriptLanguageNameBehaviour BEHAVIOUR
       DEFINED AS "The managed object instance name of a launch script represented by a
       general string.";;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx7(7)};
```

```
defaultExecutionParameterList  ATTRIBUTE

        WITH ATTRIBUTE SYNTAX CSModule.ExecutionParameterList;

        MATCHES FOR EQUALITY;

        BEHAVIOUR

        defaultExecutionParameterListBehaviour BEHAVIOUR

        DEFINED AS "A set of managed object instance names of the script instructions and
        parameter values (if required) as inputs to instances to be executed by
        default.";;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx8(8)};


executingParameters   ATTRIBUTE

        WITH ATTRIBUTE SYNTAX CSModule.ExecutionParameter;

        MATCHES FOR EQUALITY;

        BEHAVIOUR

        executingParametersBehaviour BEHAVIOUR

        DEFINED AS "A set of managed object instance names of the script instructions and
        parameter values (if required) as inputs to script executions.";;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx9(9)};


threadId   ATTRIBUTE

        WITH ATTRIBUTE SYNTAX CSModule.ThreadId;

        MATCHES FOR EQUALITY;

        BEHAVIOUR

        threadIdBehaviour   BEHAVIOUR

        DEFINED AS "The managed object instance name of a thread executing script
        instruction(s).";;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx10(10)};


triggerId ATTRIBUTE

        WITH ATTRIBUTE SYNTAX CSModule.TriggerId;

        MATCHES FOR EQUALITY;

        BEHAVIOUR

        triggerIdBehaviour BEHAVIOUR

        DEFINED AS "The managed object instance name of a trigger initiating the execution
        of a launch script.";;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx11(11)};
```

A.5 Notification definitions

```
executionResultInfo   NOTIFICATION

        WITH INFORMATION SYNTAX CSModule.ExecutionResultInfo;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) notification(10) xx1(1)};


triggerResultInfo NOTIFICATION

        WITH INFORMATION SYNTAX CSModule.TriggerResultInfo;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) notification(10) xx2(2)};
```

A.6 Action definitions

```
resume ACTION

        BEHAVIOUR resumeBehaviour BEHAVIOUR

              DEFINED AS "An action directed at a basicSpawnerClass object, causing
              unconditional resumption of all script executions by the basicSpawnerClass
              object which were initiated by a particular trigger. The value of
              controlStatus becomes empty as the result of a resume action.";;

        WITH INFORMATION SYNTAX CSModule.SpawnerObjectId;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) action(9) xx1(1)};


suspend ACTION

        BEHAVIOUR suspendBehaviour BEHAVIOUR

        DEFINED AS "An action directed at a basicSpawnerClass object, causing

        unconditional suspension of all script executions by the

        basicSpawnerClass object which were initiated by a particular trigger. The value
        of controlStatus becomes ‘suspended’ as a result of a suspend action.";;

        WITH INFORMATION SYNTAX CSModule.SpawnerObjectId;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) action(9) xx2(2)};


terminate ACTION

        BEHAVIOUR terminateBehaviour BEHAVIOUR

DEFINED AS "An action directed at a launch pad, causing unconditional termination of all
scripts by the launch pad, which was initiated by a particular trigger.";;

        WITH INFORMATION SYNTAX CSModule.TriggerId;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) action(9) xx3(3)};


trigger ACTION

        BEHAVIOUR triggerBehaviour BEHAVIOUR

        DEFINED AS "An initiator of script execution by causing a launch pad to spawn one
        or more threads.";;

        WITH INFORMATION SYNTAX CSModule.TriggerParameters;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) action(9) xx4(4)};
```

A.7 Name binding definitions

```
commandSequencer-system   NAME BINDING

        SUBORDINATE OBJECT CLASS commandSequencer AND SUBCLASSES;

        NAMED BY SUPERIOR OBJECT CLASS system AND SUBCLASSES;

        WITH ATTRIBUTE commandSequencerId;

        BEHAVIOUR csSystemContainmentBehaviour BEHAVIOUR

        DEFINED AS "Superior object class is system and subordinate object

        class is commandSequencer.";;

        CREATE WITH-REFERENCE-OBJECT, WITH-AUTOMATIC-INSTANCE-NAMING;

        DELETE ONLY-IF-NO-CONTAINED-OBJECTS;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) nameBinding(6) xx1(1)};


launchPad-commandSequencer NAME BINDING

        SUBORDINATE OBJECT CLASS launchPad AND SUBCLASSES;
        NAMED BY SUPERIOR OBJECT CLASS commandSequencer AND SUBCLASSES;
        WITH ATTRIBUTE launchPadId;
```

```
        BEHAVIOUR lpCsContainmentBehaviour BEHAVIOUR

        DEFINED AS "Naming a command sequence launch pad with respect to a command
        sequencer indicates that the command sequencer is the service provider for the
        launch pad.";;

        CREATE WITH-AUTOMATIC-INSTANCE-NAMING;

        DELETE DELETES-CONTAINED-OBJECTS;

        REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) nameBinding(6)
xx2(2)};


thread-synchronousLaunchPad  NAME BINDING

        SUBORDINATE OBJECT CLASS   thread AND SUBCLASSES;

        NAMED BY SUPERIOR OBJECT CLASS synchronousLaunchPad AND SUBCLASSES;

        WITH ATTRIBUTE   threadId;

        BEHAVIOUR threadSyncLpContainmentBehaviour BEHAVIOUR

        DEFINED AS "The superior object class synchronousLaunchPad acts as an IVMO for the
        subordinate object class thread.";;

        DELETE DELETES-CONTAINED-OBJECTS;

        REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21)

nameBinding(6) xx3(3)};


suspendableThread-asynchronousLaunchPad   NAME BINDING

        SUBORDINATE OBJECT CLASS   suspendableThread AND SUBCLASSES;

        NAMED BY SUPERIOR OBJECT CLASS asynchronousLaunchPad AND SUBCLASSES;

        WITH ATTRIBUTE   threadId;

        BEHAVIOUR threadAsyncLpContainmentBehaviour BEHAVIOUR

        DEFINED AS "The superior object class asynchronousLaunchPad acts as an IVMO for
        the subordinate object class suspendable thread.";;

        DELETE DELETES-CONTAINED-OBJECTS;

        REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21)

nameBinding(6) xx4(4)};


thread-thread   NAME BINDING

        SUBORDINATE OBJECT CLASS   thread AND SUBCLASSES;

        NAMED BY SUPERIOR OBJECT CLASS thread AND SUBCLASSES;

        WITH ATTRIBUTE threadId;

        BEHAVIOUR threadContainmentBehaviour BEHAVIOUR

        DEFINED AS "The superior object class thread acts as a spawner of the subordinate
        object class thread.";;

        DELETE DELETES-CONTAINED-OBJECTS;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) nameBinding(6) xx5(5)};


suspendableThread-suspendableThread   NAME BINDING

        SUBORDINATE OBJECT CLASS   suspendableThread AND SUBCLASSES;

        NAMED BY SUPERIOR OBJECT CLASS suspendableThread AND SUBCLASSES;

        WITH ATTRIBUTE threadId;

        BEHAVIOUR suspendableThreadContainmentBehaviour BEHAVIOUR
```

```
        DEFINED AS "The superior object class suspendable thread acts as a spawner of the
        subordinate object class suspendable thread.";;

        DELETE DELETES-CONTAINED-OBJECTS;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) nameBinding(6) xx6(6)};


launchScript-system NAME BINDING

        SUBORDINATE OBJECT CLASS launchScript AND SUBCLASSES;

        NAMED BY SUPERIOR OBJECT CLASS system AND SUBCLASSES;

        WITH ATTRIBUTE scriptId;

        BEHAVIOUR lsSystemContainmentBehaviour BEHAVIOUR

        DEFINED AS "The superior object class is system and subordinate object
        class is launchScript.";;

        CREATE WITH-REFERENCE-OBJECT, WITH-AUTOMATIC-INSTANCE-NAMING;

        DELETE ONLY-IF-NO-CONTAINED-OBJECTS;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) nameBinding(6) xx7(7)};
```

A.8 ASN.1 definitions

```
CSModule {joint-iso-itu-t ms(9) function(2) part21(21) asn1Module(2) 0}

DEFINITIONS IMPLICIT TAGS ::=

BEGIN


-- EXPORTS everything


IMPORTS

        SimpleNameType

       FROM Attribute-ASN1Module {joint-iso-itu-t ms(9) smi(3) part2(2)

asn1Module(2) 1 }

       ObjectInstance, Attribute, CMISSync, CMISFilter, ModifyOperator, Scope,

       BaseManagedObjectId FROM CMIP-1 {joint-iso-itu-t ms(9) cmip(1) modules(0)

protocol(3)}

       AE-title FROM ACSE-1 {joint-iso-itu-t association-control(2) abstract-syntax(1)

apdus(0) version(1)};


       cmdSeqRelationshipClasses OBJECT IDENTIFIER ::= {joint-iso-itu-t ms(9) function(2)
part21(21) relationshipClass(11) }

       cmdSeqRelationshipMappings OBJECT IDENTIFIER ::= {joint-iso-itu-t ms(9) function(2)
part21(21) relationshipMapping(12)}

        cmdSeqRelationshipRoles OBJECT IDENTIFIER ::= {joint-iso-itu-t ms(9) function(2)
part21(21) relationshipRole(13)}


--

-- Range Constraints used for relationship class definitions

--

RangeFromOneToOne ::= INTEGER (1 .. 1)


RangeFromZeroToMax ::= INTEGER (0 .. MAX)


--
```

```
-- Counter size constraint
--
MaxCounterSize ::= INTEGER{unlimited(0)}-- size in octets

ExecutionResultInfo ::= SEQUENCE {triggerId TriggerId,
                                  scriptId ScriptId,
                                  threadId ThreadId,
                                  errorCode ErrorCode,
                                  executionResultType ExecutionResultType,
                                  executionResult SET OF Attribute}

TriggerResultInfo ::=  SEQUENCE {triggerId TriggerId,
                          CHOICE {singleTriggerResult ResultInfoFromThread,
                                  sequentialTriggerResult SEQUENCE OF
                                                      ResultInfoFromThread,
                                  parallelTriggerResult SET OF ResultInfoFromThread}}

ResultInfoFromThread ::= SEQUENCE{executionType  ExecutionType,
                                  errorCode      ErrorCode,
                                  executionResultType ExecutionResultType,
                                  executionResult SET OF Attribute}

ExecutionType ::= CHOICE {singleExecution ScriptThreadSet,
                          parallelExecution SET OF ScriptThreadSet,
                          sequentialExecution SEQUENCE OF ScriptThreadSet}

ScriptThreadSet ::= SEQUENCE {scriptId  ScriptId,
                              threadId    ThreadId}

SpawnerObjectId ::= SEQUENCE  {triggerId  TriggerId,
                               CHOICE { threadId ThreadId,
                                        launchPadId LaunchPadId}}

ExecutionResultType ::= OBJECT IDENTIFIER
CommandSequencerId ::=  ObjectInstance
ScriptId ::=  ObjectInstance
ThreadId ::=  ObjectInstance
TriggerId ::= ObjectInstance
LaunchPadId ::= ObjectInstance

ScriptList ::= CHOICE {scriptId ScriptId,
                       sequentialScriptList SEQUENCE OF ScriptId,
                       parallelScriptList SET OF ScriptId}
AvailableScriptList ::= SET OF ScriptList
emptyScriptList AvailableScriptList ::= {}
emptyExecutionParameterList ExecutionParameterList ::= sequentialExecutionList:{}

TriggerParameters  ::= SEQUENCE {triggerId TriggerId,
                                 executionParameterList ExecutionParameterList}
```

```
ExecutionParameterList ::= CHOICE {executionParameter ExecutionParameter,
                                   sequentialExecutionList SEQUENCE OF
                                   ExecutionParameter,
                                   parallelExecutionList SET OF
                                   ExecutionParameter}

ExecutionParameter  ::= SEQUENCE  {scriptId ScriptId,
                                   scriptParameters SEQUENCE OF Attribute}

emptyParameterList   ExecutionParameterList  ::= sequentialExecutionList:{ }

ErrorCode ::= SET OF INTEGER {noError(0),
                              noScriptError(1),
                              scriptRejectedError(2),
                              invalidParameterTypeError(3),
                              invalidParameterValueError(4),
                               scriptSyntaxError(5),
                              scriptExecutionFailedError(6),
                              invalidParmeterNumber(7),
                              unauthorizedAccessError(8)}
ScriptLanguageName ::=  OBJECT IDENTIFIER
ScriptContent  ::=  GeneralString
ModificationList ::= SET OF SEQUENCE{modifyOperator  [2] IMPLICIT
                                     ModifyOperator DEFAULT replace,
                                     attributeId AttributeId,
                                     attributeValue ANY DEFINED BY
                                     attributeId  OPTIONAL
                                      -- absent for   setToDefault
                                    }


 END
```

Annex B

General Relationship Model

(This annex forms an integral part of this Recommendation | International Standard)

This following is the GRM for the command sequencer.

```
--
-- The following relationship classes support the command sequencer model
--

commandSequencer-launchPadRelationshipClassBehaviour
BEHAVIOUR DEFINED AS

       !

              The relationship class is concerned with the relationship between a command
              sequencer and its launch pads used to initiate the execution of scripts by
              means of threads requiring the support services provided by the command
              sequencer.

       !;


commandSequencer-LaunchPad-bindingBehaviour

BEHAVIOUR DEFINED AS

       !

              This notification occurs upon the binding of a launch pad into the command
              sequencer / launch pad relationship.

       !;


commandSequencer-LaunchPad-unbindingBehaviour

BEHAVIOUR DEFINED AS

       !

              This notification occurs when a launch pad is removed from  the command
              sequencer / launch pad relationship. A launch pad may be removed from the
              relationship only if its administrative state is locked.

       !;


commandSequencer-launchPad-RelationshipClass
RELATIONSHIP CLASS

       BEHAVIOUR commandSequencer-launchPadRelationshipClassBehaviour,

                 commandSequencer-LaunchPad-bindingBehaviour,

           commandSequencer-LaunchPad-unbindingBehaviour;

       SUPPORTS

              ESTABLISH,

              QUERY,

              TERMINATE,

              NOTIFY commandSequencer-LaunchPad-binding,

              NOTIFY commandSequencer-LaunchPad-unbinding;

       ROLE commandSequencerRole

              COMPATIBLE-WITH     commandSequencer

              PERMITTED-ROLE-CARDINALITY CONSTRAINT CASN1.RangeFromOneToOne

              REQUIRED-ROLE-CARDINALITY-CONSTRAINT CASN1.RangeFromOneToOne
```

```
              PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT

              CASN1.RangeFromOneToOne

              REGISTERED AS  { CSModule.cmdSeqRelationshipRoles 1 }


       ROLE    launchPadRole

              COMPATIBLE-WITH launchPad

              PERMITTED-ROLE-CARDINALITY-CONSTRAINT   CSModule.RangeFromZeroToMax

              REQUIRED-ROLE-CARDINALITY-CONSTRAINT CSModule.RangeFromZeroToMax

              BIND-SUPPORT

              UNBIND-SUPPORT

              PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT CSModule.RangeFromOneToOne

       REGISTERED AS { CSModule.cmdSeqRelationshipRoles 2 };

REGISTERED AS { CSModule.cmdSeqRelationshipClasses 1 };


commandSequencer-LaunchPad-RelationshipMapping-Behaviour

BEHAVIOUR DEFINED AS

       !

              This relationship mapping describes how the command sequencer to launch pad
              relationship class may be represented using containment. In this
              relationship mapping, the command sequencer is the superior object for the
              purposes of naming, and launch pads are contained by it. Participation in
              this relationship implies that the launch pad and its spawn may use the
              services provided by the resource represented by the command sequencer.

       !;


commandSequencer-launchPad-RelationshipMapping

RELATIONSHIP MAPPING

       RELATIONSHIP CLASS
       commandSequencer-launchPad-RelationshipClass;

       BEHAVIOUR commandSequencer-launchPad-RelationshipMapping-Behaviour;


       ROLE commandSequencerRole

              RELATED-CLASSES commandSequencer
              REPRESENTED BY NAMING

                     launchPad-commandSequencer-NameBinding
                     USING SUPERIOR,


       ROLE launchPadRole

              RELATED-CLASSES launchPad
              REPRESENTED BY NAMING

                     launchPad-commandSequencer-NameBinding
                     USING SUBORDINATE;


       OPERATIONS  MAPPING

              ESTABLISH    MAPS-TO-OPERATION

                           CREATE commandSequencer OF commandSequencerRole

              TERMINATE    MAPS-TO-OPERATION

                           DELETE commandSequencer OF commandSequencerRole
```

```
            NOTIFY commandSequencer-launchPad-binding

                        MAPS-TO-OPERATION

                        NOTIFICATION "CCITT Rec. X.721 | ISO/IEC 10165-2:1992":
                        objectCreationNotification

                              OF commandSequencerRole

            NOTIFY commandSequencer-launchPad-unbinding

                        MAPS-TO-OPERATION

                        NOTIFICATION "CCITT Rec. X.721 | ISO/IEC 10165-2:1992":
                        objectDeletionNotification

                              OF commandSequencerRole

            BIND   MAPS-TO-OPERATION

                        CREATE launchPad OF launchPadRole

            UNBIND       MAPS-TO-OPERATION

                        DELETE launchPad

                              OF launchPadRole

            QUERY  MAPS-TO-OPERATION

                        GET OF commandSequencerRole

                        GET OF launchPadRole;

REGISTERED AS {CSModule.cmdSeqRelationshipMappings 1}


scriptReferenceRelationshipClassBehaviour

BEHAVIOUR DEFINED AS

      !

            This relationship class describes the relationship existing between scripts
            and an object which references them for the purposes of identifying task(s)
            to be carried out.

      !;


scriptReferencerRelationshipClass

RELATIONSHIP CLASS

      BEHAVIOUR scriptReferencerRelationshipClassBehaviour;
      SUPPORTS

            ESTABLISH,
            TERMINATE,
            QUERY;


      ROLE scriptUserRole

            COMPATIBLE-WITH scriptReferencer

            PERMITTED-ROLE-CARDINALITY-CONSTRAINT CSModule.RangeFromOneToOne

            REQUIRED-ROLE-CARDINALITY CONSTRAINT CSModule.RangeFromOneToOne

            PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT

                  CSModule.RangeFromZeroToMax

      REGISTERED AS {CSModule.cmdSeqRelationshipRoles 3}


      ROLE scriptRole

            COMPATIBLE-WITH launchScript

            PERMITTED-ROLE-CARDINALITY-CONSTRAINT CSModule.RangeFromOneToOne

            REQUIRED-ROLE-CARDINALITY-CONSTRAINT CSModule.RangeFromOneToOne
```

```
            PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT

                  CSModule.RangeFromZeroToMax

      REGISTERED AS {CSModule.cmdSeqRelationshipRoles 4}


REGISTERED AS {CSModule.cmdSeqRelationshipClasses 2}


launchPad-launchScriptRelationshipMappingBehaviour BEHAVIOUR

DEFINED AS

!

      This relationship mapping describes the relationship between the launch pad and the
      launch script. The launch pad initiates execution of the launch script and
      references it for the purposes of execution.

!;


launchPad-LaunchScriptMapping
RELATIONSHIP MAPPING

       RELATIONSHIP CLASS scriptReferenceRelationshipClass;

       BEHAVIOUR launchPad-launchScriptMappingBehaviour;

       ROLE scriptUserRole

           RELATED CLASSES launchPad

           REPRESENTED BY ATTRIBUTE scriptList

       ROLE   scriptRole

           RELATED CLASSES launchScript;

       OPERATIONS MAPPING

           ESTABLISH MAPS-TO-OPERATION

               CREATE launchPad OF scriptUserRole

               -- using SET-BY-CREATE of scriptList --

               REPLACE scriptList of scriptUserRole

               -- which effectively adds a scriptId to scriptList --

               ADD scriptList of scriptUserRole,

           TERMINATE MAPS-TO-OPERATION

               DELETE launchPad OF scriptUserRole

               REPLACE scriptList OF scriptUserRole

               -- which effectively removes a scriptId from scriptList --

               REMOVE scriptList OF scriptUserRole,

           QUERY MAPS-TO-OPERATION

               GET scriptList OF scriptUserRole;

REGISTERED AS {CSModule.cmdSeqRelationshipMappings 2};


thread-launchScriptRelationshipMappingBehaviour BEHAVIOUR

DEFINED AS

!
      This relationship class describes the relationship between a thread and launch
      script. The launch script is referenced by the thread by means of the scriptId
      attribute from the scriptIds in the scriptList.

!;


thread-launchScriptMapping
RELATIONSHIP MAPPING
```

```
        RELATIONSHIP CLASS scriptReferenceRelationshipClass;

        BEHAVIOUR thread-launchScriptRelationshipMappingBehaviour;

        ROLE scriptUserRole

            RELATED CLASSES thread

            REPRESENTED BY ATTRIBUTE scriptId

            QUALIFIED BY scriptList

        ROLE   scriptRole

            RELATED CLASSES launchScript;

        OPERATIONS MAPPING

            ESTABLISH MAPS-TO-OPERATION

                CREATE OF scriptUserRole,

            TERMINATE MAPS-TO-OPERATION

                DELETE OF scriptUserRole,

            QUERY MAPS-TO-OPERATION

                GET scriptId OF scriptUserRole;

REGISTERED AS {CSModule.cmdSeqRelationshipMappings 3};


spawner-progeny-RelationshipClass

RELATIONSHIP CLASS

        BEHAVIOUR spawner-progenyRelationshipClassBehaviour BEHAVIOUR

       DEFINED AS

              !

              When an instance of this class causes the creation of new objects, it serves
              as an IVMO during the creation by supplying values for the new object's
              attributes based on its own defaultExecutionParameterList attribute and any
              parameters which were supplied to it as part of the action or local
              Instances of this class are capable of causing the creation of new object
              instances. The newly created instances will be contained by this instance,
              and their names will be automatically generated. Until all created objects
              are complete, this object's usage status will be "in use". If this object's
              administrative state is "locked" such new objects cannot be created. If, due
              to local resource limitations, this object is incapable of supporting more
              contained objects, its usage status will be "busy".


              An instance of this class may cause the creation of new object instances
              from a single scriptId, from a set of scriptIds in any order or from a
              sequence of scriptIds in the order specified in the list i.e. after the
              first has been created the second may not be created until after the first
              is completed, and so on. The value of the created object's scriptId
              attribute gets its value from the corresponding element of this object's
              scriptList.

            !;

        SUPPORTS

            ESTABLISH

            TERMINATE;


        ROLE spawnerRole COMPATIBLE-WITH basicSpawnerClass

            PERMITTED-ROLE-CARDINALITY-CONSTRAINT CSModule.RangeOneToOne

            REQUIRED-ROLE-CARDINALITY-CONSTRAINT CSModule.RangeOneToOne
```

```
            PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT

                CSModule.RangeOneToOne

            REGISTERED AS {CSModule.cmdSeqRoles 5}

        ROLE progenyRole COMPATIBLE-WITH thread

            PERMITTED-ROLE-CARDINALITY-CONSTRAINT CSModule.RangeZeroToMax

            REQUIRED-ROLE-CARDINALITY-CONSTRAINT CSModule.RangeZeroToMax

            PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT
        CSModule.RangeOneToOne

            REGISTERED AS {CSModule.cmdSeqRoles 6}

REGISTERED AS {CSModule.cmdSeqRelationshipClasses 3};
```

Annex C

Management Information Definitions for Event Discrimination Counting

(This annex does not form an integral part of this Recommendation | International Standard)

The Event Discrimination Counter (EDC) class object counts the number of input events. Before counting up, the EDC tests values of attributes related to the event or an object generating the event and discriminates the event.

C.1 Managed object class

```
eventDiscriminationCounter MANAGED OBJECT CLASS

DERIVED FROM   "DMI":discriminator;

CHARACTERIZED BY

edcPackage PACKAGE

        BEHAVIOUR edcBehaviour BEHAVIOUR

       DEFINED AS

       "If the result of discrimination of a potential event report evaluates to TRUE and
       the event discrimination counter is in the Unlocked and Enabled state and does not
       exhibit the off-duty availability status, then the counter value of the counter
       attribute is incremented.";;

              ATTRIBUTES

                    "CCITT Rec. 721 | ISO/IEC 10165-2:1992":counter GET,

                    maxCounterSize GET;

              NOTIFICATIONS

                    "CCITT Rec. 721 | ISO/IEC10162:1992":processingErrorAlarm;;;

CONDITIONAL PACKAGES

       counterAlarmPackage PRESENT IF "a counter is of finite size and a notification is
       triggered by a capacity alarm threshold.";

REGISTERED AS {joint-iso-itu-t ms(9) ms(9) function(2) part21(21)

managedObjectClass(3) xx11(11)};
```

C.2 Package

```
counterAlarmPackage PACKAGE

        BEHAVIOUR

        counterAlarmBehaviour BEHAVIOUR

        DEFINED AS "When the counter value reaches the capacity alarm threshold(as a
                percentage of maximum counter size), the EDC(Event Discrimination Counter)
                generates an event indicating that a capacity threshold has been reached
                or exceeded. In reporting the capacity threshold event, use is made of the
                alarm report defined in CCITT Rec. X.733 | ISO/IEC 10164-4. Only the
                following parameters of the alarm report shall be used and all parameters
                are mandatory when used for reporting counter capacity threshold alarms.

                 Managed Object Class - This parameter shall identify the

                 counter class.

                 Managed Object Instance - This parameter shall identify the

                 instance of the counter that generated the event.

                Alarm Type - This parameter shall indicate that a processing

                           error alarm has occurred.
```

```
            Event time - This parameter carries the time at

             which the capacity threshold event occurred.

             Perceived Severity - This parameter will indicate the severity

            assigned to the capacity threshold event. When the 100%

            counter full condition is reached, a severity value of critical

            shall be assigned to this event.

             Monitored Attributes - This parameter shall carry the maximum  counter size
            attribute of the EDC.

             Probable Cause - This parameter shall carry the value congestion.

             Threshold Info - This parameter shall carry the capacity threshold

            value (as percentage of total capacity) that was reached or exceeded in
            generating this event.";;

           ATTRIBUTES

             "CCITT Rec. 721| ISO/IEC 10165-2:1992":capacityAlarmThreshold GET-REPLACE
ADD-REMOVE;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx15(15)};
```

C.3 Attribute

```
maxCounterSize ATTRIBUTE

        WITH ATTRIBUTE SYNTAX CSModule.MaxCounterSize;

        MATCHES FOR EQUALITY, ORDERING;

        BEHAVIOUR

        maxSizeOrderingBehaviour BEHAVIOUR

        DEFINED AS "This Attribute represent the largest value of the counter. The
        ordering in the same as for sequentially increasing positive integers except that
        a value of zero is largest and denotes infinite size.";;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx12(12)};
```

Annex D

cmisScript Management Support Object Class

(This annex forms an integral part of this Recommendation | International Standard)

The following is a script managed object class, CMIS script, which can be defined to handle the invocation of CMIS operations.

D.1 Attributes

D.1.1 Attributes imported from the definition of management information

This Specification references the following management attributes, whose abstract syntax is specified in CCITT Rec. X.721 | ISO/IEC 10165-2:

a) attributeIdentifierList;

b) objectClass;

c) attributeList.

This Specification references the following management attributes, whose abstract syntax is specified in ITU-T Rec. X.711 | ISO/IEC 9596-1:

a) synchronization;

b) scope;

c) filter;

d) baseManagedObjectId;

e) modificationList.

D.2 Definitions

D.2.1 **cmisScript:** A management support object, which directs the invocation of a single CMIS operation.

Five types CMIS scripts are specified:

D.2.2 **getCmisScript:** A CMIS script that represents a GET operation.

D.2.3 **set CmisScript:** A CMIS script that represents a SET operation.

D.2.4 **actionCmisScript:** A CMIS script that represents an ACTION operation.

D.2.5 **createCmisScript:** A CMIS script that represents a CREATE operation.

D.2.6 **deleteCmisScript:** A CMIS script that represents a DELETE operation.

CMIS scripts are single-instruction scripts which give operational details to threads. For example, the script languages may refer to appropriate command records in order to make a CMIS requests to an agent.

D.3 getCmisScript

D.3.1 Characteristics of getCmisScript

The following attributes are defined in the getCmisScript:

– baseManagedObjectId;

– synchronization;

– scope;

– filter;

– attributeIdentifierList.

D.3.2 Packages of getCmisScript

getCmisScript has the following mandatory package:

- getCmisScriptPackage.

D.4 setCmisScript

D.4.1 Characteristics of setCmisScript

setCmisScript has the following attributes definitions:

- baseManagedObjectId;
- synchronization;
- scope;
- filter;
- modificationList.

D.4.2 Packages of setCmisScript

The setCmisScript has the following mandatory package:

- setCmisScriptPackage.

D.5 actionCmisScript

D.5.1 Characteristics of actionCmisScript

The actionCmisScript has the following attribute definitions:

- baseManagedObjectId;
- synchronization;
- scope;
- filter.

D.5.2 Packages of the actionCmisScript

The action command record class has the following mandatory package:

- actionCmisScriptPackage.

D.6 createCmisScript

D.6.1 Characteristics of createCmisScript

The action command record has the following attribute definitions:

- objectClass;
- attributeList.

D.6.2 Packages of createCmisScript

The create command record class has the following mandatory package:

- createCmisScriptPackage.

It has the following conditional packages:

- managedObjectInstancePackage;
- superiorObjectInstancePackage;
- referenceObjectInstancePackage.

D.7 deleteCmisScript

D.7.1 Characteristics of the deleteCmisScript

The deleteCmisScript has the following attribute definitions:

- baseManagedObjectId;
- synchronization;
- scope;
- filter.

D.7.2 Packages of the deleteCmisScript

The deleteCmisScript has the following mandatory package:

- deleteCmisScriptPackage.

D.8 Services

D.8.1 Get reporting service definition

This clause specifies the get reporting service, and maps it onto the CMIS M-EVENT-REPORT services.

D.8.2 Set reporting service definition

This clause specifies the set reporting service, and maps it onto the CMIS M-EVENT-REPORT services.

D.8.3 Action reporting service definition

This clause specifies the action reporting service, and maps it onto the CMIS M-EVENT-REPORT services.

D.8.4 Creation reporting service definition

This service allows an MIS-user, in agent role, to report the creation of a managed object. It is defined as both a confirmed and as a non-confirmed service and mapped onto the CMIS M-EVENT-REPORT services. This service is defined in CCITT Rec. X.730 | ISO/IEC 10164-1.

D.8.5 Deletion report service definition

This service allows an MIS-user, in agent role, to report the deletion of a managed object. It is defined as both a confirmed and as a non-confirmed service and mapped onto the CMIS M-EVENT-REPORT services. This service is defined in CCITT Rec. X.730 | ISO/IEC 10164-1.

D.9 GDMO template

D.9.1 Managed object definitions

```
cmisScript MANAGED OBJECT CLASS
       DERIVED FROM launchScript;
CHARACTERIZED BY cmisScriptPackage PACKAGE
       BEHAVIOUR
             cmisScriptBehaviour BEHAVIOUR
       DEFINED AS
       !
             An instance of this managed object class models information necessary to
             execute a single CMIS operation.
       !;;;;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3)
xx12(12)};
getCmisScript MANAGED OBJECT CLASS
       DERIVED FROM cmisScript;
CHARACTERIZED BY getCmisScriptPackage PACKAGE
       BEHAVIOUR
```

```
        getCmisScriptBehaviour   BEHAVIOUR
        DEFINED AS
        !
                An instance of this managed object class models information
                necessary to execute a single CMIS GET operation.
        !;;
        ATTRIBUTES
                baseManagedObjectId GET,
                synchronization GET-REPLACE,
                scopeGET-REPLACE,
                filter GET-REPLACE,
                "CCITT Rec. X.721 | ISO/IEC 10165-2:1992":attributeIdentifierList
                 GET-REPLACE ADD-REMOVE;;;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3)
xx13(13)};


setCmisScript MANAGED OBJECT CLASS
        DERIVED FROM cmisScript;
CHARACTERIZED BY setCmisScriptPackage PACKAGE
         BEHAVIOUR
        setCmisScriptBehaviour BEHAVIOUR
        DEFINED AS
        !
                An instance of this managed object class models information necessary to
                execute a single CMIS SET operation.
        !;;
        ATTRIBUTES
                baseManagedObjectId GET,
                synchronization      GET-REPLACE,
                scope                GET-REPLACE,
                filter               GET-REPLACE,
                modificationList     GET-REPLACE ADD-REMOVE;;;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3)
xx14(14)};


actionCmisScript MANAGED OBJECT CLASS
        DERIVED FROM cmisScript;
        CHARACTERIZED BY
actionCmisScriptPackage PACKAGE BEHAVIOUR
                actionCmisScriptBehaviour BEHAVIOUR
                DEFINED AS
                !
                An instance of this managed object class models information necessary to
                execute a single CMIS ACTION operation.
                !;;
```

```
        ATTRIBUTES

                baseManagedObjectId GET,

                synchronization     GET-REPLACE,

                scope               GET-REPLACE,

                filter              GET-REPLACE;;;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3)
xx15(15)};



createCmisScript  MANAGED OBJECT CLASS
DERIVED FROM cmisScript;
CHARACTERIZED BY

       createCmisScriptPackage PACKAGE

       BEHAVIOUR

              createCmisScriptBehaviour BEHAVIOUR

              DEFINED AS

              !

                     An instance of this managed object class models information necessary
                     to execute a single CMIS CREATE operation.

              !;;

       ATTRIBUTES

              "CCITT Rec. 721 | ISO/IEC 10165-2:1992": objectClass  GET,

              "CCITT Rec. 721 | ISO/IEC 10165-2:1992": attributeList GET-REPLACE ADD-
              REMOVE;;;

       CONDITIONAL PACKAGES

              managedObjectInstancePackage

                     PRESENT IF "the superiorObjectInstancePackage is not present.",

              superiorObjectInstancePackage

                     PRESENT IF "the managedObjectInstance Package is not present.",

              referenceObjectInstancePackage

                     PRESENT IF "the manager has the specified value.";

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3)
xx16(16)};



deleteCmisScript  MANAGED OBJECT CLASS
DERIVED FROM  cmisScript;
CHARACTERIZED BY

       deleteCmisScriptPackage PACKAGE

       BEHAVIOUR

       deleteCmisScriptBehaviour BEHAVIOUR
       DEFINED AS

       !

              An instance of this managed object class models information  necessary to
              execute a single CMIS DELETE operation.

       !;;
       ATTRIBUTES

              baseManagedObjectId GET,
              synchronization     GET-REPLACE,
              scope         GET-REPLACE,
              filter              GET-REPLACE;;;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3)
xx17(17)};
```

D.9.2 Package definitions

```
 managedObjectInstancePackage   PACKAGE
  ATTRIBUTES
        managedObjectInstance   GET-REPLACE;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx16(16)};


superiorObjectInstancePackage   PACKAGE
        ATTRIBUTES
        superiorObjectInstance   GET-REPLACE;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx17(17)};


       referenceObjectInstancePackage  PACKAGE
        ATTRIBUTES
        referenceObjectInstance  GET-REPLACE;
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx18(18)};
```

D.9.3 Attribute definitions

```
baseManagedObjectId   ATTRIBUTE
        WITH ATTRIBUTE SYNTAX CSModule.BaseManagedObjectId;
        MATCHES FOR EQUALITY;
        BEHAVIOUR baseManagedObjectIdBehaviour
        BEHAVIOUR
        DEFINED AS
       !
           This is the identifier for the information on the CMIS
             operation to be executed.
        !;;
        REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx13(13)
};


scope ATTRIBUTE
        WITH ATTRIBUTE SYNTAX CSModule.Scope;
        MATCHES FOR EQUALITY;
        BEHAVIOUR
        scopeBehaviour     BEHAVIOUR
        DEFINED AS
        !
              This is the first phase in the selection of managed object(s) to   which the
              CMIS script operations should be directed. It indicates the managed
              object(s) to which a filter should be applied.
        !;;
         REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7)
xx14(14)};


filter ATTRIBUTE
        WITH ATTRIBUTE SYNTAX CSModule.CMISFilter;
        MATCHES FOR EQUALITY;
```

```
        BEHAVIOUR
        filterBehaviour    BEHAVIOUR
        DEFINED AS
        !
              This is the second phase in the selection of managed object(s) to   which
              the CMIS script operations should be directed. A set of tests is applied to
              each of the previously scoped managed objects to extract a subset.
        !;;
        REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx15(15)
};


synchronization ATTRIBUTE
       WITH ATTRIBUTE SYNTAX CSModule.CMISSync;
        MATCHES FOR EQUALITY;
        BEHAVIOUR
        synchronizationBehaviour BEHAVIOUR
        DEFINED AS
        !
              This indicates the manner in which operations are to be synchronized across
              managed object instances when multiple managed objects have been selected by
              the scoping and filtering mechanisms.
        !;;
        REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx16(16)
};

modificationList ATTRIBUTE
       WITH ATTRIBUTE SYNTAX CSModule.ModificationList;
        MATCHES FOR EQUALITY;
        BEHAVIOUR
        modificationListBehaviour BEHAVIOUR
        DEFINED AS
        !
              This represents the list of attributes to be modified by the CMISSet script
              and contains the values to which these attributes should be set.”;;


        !;;
        REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx17(17)
};

superiorObjectInstance ATTRIBUTE
        WITH ATTRIBUTE SYNTAX CSModule.ObjectInstance;
        MATCHES FOR EQUALITY;
        BEHAVIOUR
        superiorObjectInstanceBehaviour BEHAVIOUR
        DEFINED AS
        !
            This attribute identifies the existing managed object instance
            which is supplied, the managedObjectInstance attribute shall not
            be supplied.
        !;;
```

```
REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx18(18)};



referenceObjectInstance ATTRIBUTE

        WITH ATTRIBUTE SYNTAX CSModule.ObjectInstance;

        MATCHES FOR EQUALITY;

        BEHAVIOUR

        referenceObjectInstanceBehaviour BEHAVIOUR

        DEFINED AS

        !

            The managed object instance name of the same class as the managed object to be
            created. Attribute values associated with the reference object instance become
            default values for those not specified by the attribute list attribute.



        !;;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx19(19)};
```

Annex E

CMIP_CS managed object class

(This annex does not form an integral part of this Recommendation | International Standard)

The following is an example of how additional subclasses may be defined in order to contain attributes such as AE title.

E.1 cmipCS

E.1.1 Overview

- Subclass of cmip protocol machine and command sequencer.
- Defines the calling AE title for the command sequencer.

E.1.2 Characteristics of the cmipCS managed object class

- aetitle.

E.1.3 Packages of cmipCS

The cmipCS managed object class has the following mandatory package:

- aetitle package.

E.1.4 GDMO definitions

```
cmipCS MANAGED OBJECT CLASS

DERIVED FROM commandSequencer;

CHARACTERIZED BY aeTitlePackage;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3)
xx18(18)};


aeTitlePackage PACKAGE

ATTRIBUTES

           aetitle GET;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx19(19)};
```

E.1.5 Attribute definitions

```
aetitle ATTRIBUTE

       WITH ATTRIBUTE SYNTAX CSModule.AE-title;

       MATCHES FOR EQUALITY;

       BEHAVIOUR

       aeTitleBehaviour BEHAVIOUR

      DEFINED AS "An instance of this managed object class defines the

      calling AE title for the command sequencer";;

REGISTERED AS {joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx20(20)};
```

Annex F

Systems Management Scripting Language (SMSL)

(This annex forms an integral part of this Recommendation | International Standard)

This annex defines a general string scripting language, SMSL, for writing procedures for execution in a command sequencer environment.

SMSL has been provided with functions needed for this environment. To accomplish this, SMSL sacrifices some of the completeness of languages such as C, csh, Perl, or awk while implementing some of the statements and functions that make those languages so powerful and popular.

F.1 Mapping GDMO onto SMSL

SMSL has a virtual machine which is capable of interpreting GDMO as described in ISO/IEC 10165-Series and 10164-Series, which has been mapped into the SMSL data types in accordance with the SMSL object model as described in F.14.11. Scripts written in SMSL are interpreted and executed by the SMSL virtual machine. In the command sequencer environment, the SMSL virtual machine acts in the manager role for systems management purposes. The SMSL virtual machine should be able to interpret all the SMSL data types which are used by SMSL scripts, i.e. all the types used in the script without the definitions of these types are known implicitly by the virtual machine, from the viewpoint of the script writer. The SMSL virtual machine should be able to interpret all the SMSL instances which have already been instantiated, i.e. all the references to previously instantiated objects are known to the virtual machine, from the viewpoint of the script writer.

F.2 SMSL Built-in functions

SMSL includes a number of built-in functions for creating and manipulating objects and general-purpose functions such as mathematical, logical, and I/O functions. Following is a summary of the SMSL built-in functions. The functions are individually described below.

F.2.1 Functions for concurrency control

SMSL includes two built-in functions for enforcing concurrency control: lock() and unlock(). These functions are typically used to linearize accesses to shared data structures and resources.

All SMSL processes attempting to linearize accesses to a resource must cooperate by requesting locks of a given lock name. All resource accesses, including the set() and get() functions, are denied shared resource access without a lock. It is the responsibility of each SMSL process to access a resource only when it holds the required lock.

F.3 Set functions for SMSL lists

SMSL includes the following functions for performing set operations on SMSL lists;

- difference() to return the list of different elements between lists;
- intersection() to return a list of elements common between lists;
- sort() to return a list in ascending or descending element order;
- subset() to verify that one list is contained within another;
- union() to return a list that is a combination of lists;
- unique() to return a list of elements that appear in only one list.

These functions process SMSL lists as sets of elements. Each member of a list is text string that ends with a space character.

The NULL set [""] is the equivalent of the null or empty set (Ø) in set theory. The NULL set is treated by the SMSL set functions as a proper set that contains no elements. The NULL string [ ] is a SMSL list element with no characters. The SMSL set functions allow lists to contain NULL strings.

The SMSL concept of a set is not the unique list of ascending or descending elements familiar to set theory. In many cases, the SMSL lists contain duplicate elements arranged in no particular order. A SMSL list can be transformed into an ordered set using the unique() function to remove duplicates and the sort() function to arrange the elements in ascending or descending order.

F.4 SMSL mathematical functions

SMSL supports a basic subset of mathematical functions. These functions are all similar to C-mathematical functions.

The SMSL mathematical functions include some run-time error checking for range and domain. Both conditions result in a run-time error message that sets the SMSL errno variable to an appropriate value.

Additionally, any nonnumeric values produced by printing the result of the function call, such as Nan or -Inf are converted to 0.0 to prevent the return value from being interpreted by SMSL as a non-numeric character string. The SMSL function also returns a run-time error message when it performs the conversion.

F.5 SMSL process synchronization

SMSL provides process synchronization within SMSL processes of a single command sequencer through condition variable primitives used with SMSL locks. These primitives are similar to constructs provided for multithreaded programming in the C-programming language on many non-threading operating systems.

F.6 SMSL shared global channels

SMSL supports the use of shared global channels for communication between a process and another process or file.

The SMSL allows one SMSL process to open a channel to an external process in a explicitly shared mode, which allows any number of other SMSL processes to send data to, and receive data from, the channel.

The ability to share channels requires that SMSL also provide a mechanism for concurrent programming techniques. SMSL provides these in the form of external synchronization primitives.

The primitives are preferable to building concurrency into the channel opening, reading, writing, and closing functions. For example, having each SMSL process lock a shared channel explicitly prevents concurrent reading by one process and writing by another. In addition, having the synchronization primitives separate from channels allows them to be used to synchronize the use of any shared resource such as the agent’s internal symbol table or an external file.

The read(), readln() and write() functions for shared channels will fail immediately (without blocking) if another SMSL process is already blocked on the channel.

F.6.1 The effect of SMSL shared global channel mechanisms

The SMSL functions ensure that all operations on a channel are serialized, with all SMSL function calls appearing to be atomic. The SMSL programmer can be assured that file channel reads and writes in different processes will take place atomically. The locks provided in SMSL prevent unpredictable interleaving of sequences of SMSL read and write calls to the channel.

The single exception to serialization on channels created using the popen() function is the allowance for a concurrent read and write operation. A read can occur when a write is pending on the channel, and a write can occur when a read is blocked or pending on the channel – thus, both a reader and a writer SMSL process can be blocked on a shared channel.

File channels opened using the fopen() function can never cause a SMSL read() or write() function to block. To enforce serialization, the second reader process cannot be blocked, nor can the second writer process be blocked; hence, the second SMSL read(), readln() and write() function on a file channel will fail.

F.7 SMSL data types and objects

SMSL has four basic data types: integer, float, string, and list. The values of all four types simple types may be manipulated as though they were character strings. Complex data types can be built up using struct and array constructs. All ASN.1 syntax types are supported by SMSL.

F.7.1 Conversion between GDMO and ASN.1 type and SMSL script-oriented type

Table F.1 shows the relationship between a GDMO and ASN.1 type and the corresponding SMSL type.

Table F.1 – Value conversion between ASN.1 and script-oriented type

Value type group	GDMO and ASN.1 type	script-oriented type
Integer group	INTEGER, BOOLEAN	integer
Float group	REAL	float
String	General string	string
Set group	SEQUENCE, SET	object type
Array expression	SEQUENCE OF, SET OF	list

Variables and values are interpreted as either strings or numbers, whichever is appropriate to the context.

A scalar (integer or float) is interpreted as true in the Boolean sense if it is not the null string or 0. Booleans returned by operators are 1 for true and 0 or “” (the null string) for false.

F.7.2 Numeric constants

Although the internal representation of an integer or floating-point constant is a string, these constants need not appear inside quotation marks in SMSL scripts. Some examples of SMSL integer and floating point constants are:

x = 3; pi = 3.14159;

Table F.2 – Examples of SMSL data types

Data type	Example representation	SMSL representation
integer	3	"3"
float	4.5	"4.5"
string	“abc”	"abc"
list	[1,3,5]	"1\n3\n5"
NOTE – “\n” is the new-line character.		

F.7.3 Complex data types

A struct is a collection of data types which can be grouped together for the purpose of representing complex ASN.1 structures such as ASN.1 SETs and SEQUENCEs can be constructed using struct. Individual struct elements may be accessed using the “.” operator.

Array expressions can be expressed using the list type in SMSL. The name binding is mapped to the value of the object name.

F.8 SMSL variables

Variables of any type can be used as lvalues – that is, they can be assigned to. As all data types are treated as strings internally, they all share a common name space. Therefore, you cannot use the same name for a scalar variable, a string variable, and a list variable.

Case is significant. “FOO”, “Foo”, and “foo” are all different names. Names must start with a letter or an underscore but can contain digits and underscores (“_”).

Some identifiers have predefined meanings. Reserved keywords – such as if and foreach – cannot be used as identifiers. Keywords are recognized as either all lowercase or all uppercase letters. In addition, predefined constants cannot be used as identifiers.

F.8.1 Default initialization of SMSL variables

SMSL does not make use of the concept of "declarations" for variables. The first appearance of an identifier serves to add it to the list of global variables for a SMSL script. All variables are initialized with a null string value each time a SMSL script is executed. This value does not change until the variable's value is defined by some explicit operation, such as assignment.

This default initialization to the empty string allows a variable to be treated as an initially empty list/string or as a numeric variable with a 0 value (since arithmetic operators treat the null string as equivalent to 0). However, reliance on this initial value causes a SMSL run-time warning message at its first use (if run-time warnings are enabled). It is considered better style to initially assign a value of "" or 0 to a list/string variable or numeric variable, respectively.

F.9 SMSL predefined constants

A number of identifiers are predefined as constants so that they can be used without needing declaration. These constants are read-only and should not be assigned to.

Table F.3 – SMSL predefined constants

Constant	Definition
ALARM	ALARM object state
OK	OK object state
OFFLINE	OFFLINE object state
VOID	VOID object state
EOF	End-of-file condition constant
true/TRUE/True yes/YES/Yes	Boolean true value (1)
false/FALSE/False no/NO/No	Boolean false value (0)

F.10 SMSL string literals

String literals are delimited by double quotation marks. String literals can be multiline, causing the new-line characters to become part of the string.

The backslash rules apply for escaping characters (such as the backslash or the quotation mark) and for making characters such as new-line or tab. These are the only string literals currently supported in SMSL.

Table F.4 – SMSL string literals

Constant	Definition
\t	tab
\n	new-line
\r	return
\b	backspace
\A . . . \Z	Ctrl-A . . . Ctrl-Z

Control characters can be embedded in SMSL string constants using \A through to \Z to represent Ctrl-A through to Ctrl-Z. A capitalized letter must always be used; lowercase letters other than those already defined (that is, t, n, r, or b) are not valid as escapes and will generate a SMSL compilation warning.

F.11 SMSL lists

List values are denoted by separating individual values with commas and by enclosing the list in square brackets:

[1, 3, 5]

The list is interpolated into a double-quoted string whose elements are separated by spaces. The list is represented internally as:

"1 3 5"

F.12 SMSL simple statements

The most common simple statement is an expression evaluated for its side effects, which is called an expression statement. The most common expression statement is an assignment operation or a function call. Every expression statement must be terminated with a semicolon:

```
y = x + 10;          # assignment
set("value",50);     # function call
s = trim(s,"\t");    # both assignment and function call
```

F.13 SMSL operators

F.13.1 Arithmetic operators

For arithmetic operators, an operand is considered a number if its first character is a digit or a minus sign (–). Otherwise, it is considered a string and converted to 0 for an empty string or 1 for a non-empty string.

The use of a non-number in an arithmetic context may result in a run-time warning.

Table F.5 – SMSL arithmetic operators

Operator	Definition
+	addition
–	subtraction
/	division
*	multiplication
%	modulus

F.13.2 Assignment operators

Following are the assignment operators for SMSL.

For example, a+=b is equivalent to a=a+b.

Table F.6 – SMSL assignment operators

Operator	Definition
=	assignment
.=	self-concatenation for strings
+=	self-addition
–=	self-subtraction
/=	self-division
*=	self-multiplication
%=	self-modulus

Bitwise assignment:

&=	self-bitwise AND
|=	self-bitwise OR
^=	exclusive OR bitwise assignment

Shift assignment:

<<=	shift left assignment
>>=	shift right assignment

Increment/Decrement operators

For example, a++ is equivalent to a=a+1.

Table F.7 – SMSL increment/decrement operators

Operator	Definition
++	increment
--	decrement

F.13.3 Bitwise operators

Following are the bitwise operators defined for SMSL.

For example, a&=b is equivalent to a=a&b.

Table F.8 – SMSL bitwise operators

Operator	Definition
&	bitwise AND
|	bitwise OR
&=	self-bitwise AND
|=	self-bitwise OR
^	exclusive OR bitwise
^=	exclusive OR bitwise assignment

F.13.4 Logical operators

The SMSL logical operators assume for their operands that true is represented by 1 or a non-empty string. False is represented by 0 or an empty string. However, when they return results, they always use 1 for true and 0 for false.

Table F.9 – SMSL logical operators

Operator	Definition
&&	logical AND
||	logical OR
!	logical NOT

F.13.5 Relational operators

The relational operators perform numeric comparisons if both operands are numbers. Otherwise they perform string comparisons (that is, lexical, dictionary ordering). A string is considered a number if it consists of only digits, the minus sign, or a period. No white space is allowed. SMSL relational operators do not consider constants in exponential notation (such as 2.3e+27) to be numbers.

F.13.6 Shift operators

The shift operators perform bit shifting within bytes.

Table F.10 – SMSL relational operators

Operator	Definition
==	equal to
!=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

Table F.11 – SMSL shift operators

Operator	Definition
<<	shift left
<<=	shift left assignment
>>	shift right
>>=	shift right assignment

F.13.7 String operators

SMSL has several operators for string and list manipulation.

[s1, s2, ...]

The list operator builds a list by joining all elements in a comma-separated list in a double-quoted string of items delimited by a space, which is SMSL's representation for lists/arrays.

=~ (equal tilde)

The =~ operator is used in the expression string =~ pattern and returns:

- 1 if the regular expression pattern is contained in string;
- 0 if the regular expression pattern is not contained in string.

If pattern is invalid, SMSL returns a run-time error message and the =~ operation returns 0 (pattern not contained).

!~ (tilde)

The !~ operator is used in the expression string !~ pattern and returns:

- 1 if the regular expression pattern is not contained in string;
- 0 if the regular expression pattern is contained in string.

If pattern is invalid, SMSL returns a run-time error message and the !~ operation returns 0 (pattern contained).

F.13.8 SMSL operator precedence and associativity

The precedence and associativity of SMSL operators is almost identical to that of C and Perl. In addition to the standard operators, there are new string operators – "." and [x,y,...] – with their associated precedences.

In Table F.12, the operators are listed in ascending order of precedence:

Table F.12 – SMSL operator precedence and associativity

Operator precedence	Associativity
=	lowest right
+=, –=, <<=, >>=, ^=	right
*=, /=, %=	right
|=, &=	right
||	left
&&	left
|	left
^	left
&	left
!=, ==, =~, !~	left
<, <=, >, >=	left
<<, >>	left
+, – (binary)	left
*, /, %	left
. (string concat)	left
!, –, ++, --	right
()	left
[] highest	left

F.14 The SMSL core scripting language

A SMSL script consists of a sequence of commands. All uninitialized user-created objects are assumed to start with a NULL or 0 value until they are defined by some explicit operation such as assignment.

SMSL is, for the most part, a free-form language. That is, lines don't have to start or end at or before a particular column; they can just continue on the next line. White space is ignored except for the separation of tokens. Comments are indicated by the # character and extend to the end of the line. For example, here is a comment about an assignment statement:

```
x = y;          # Assign the value of y to the variable x
```

F.14.1 SMSL compound statements

SMSL compound statements include loop statements and if statements. In SMSL, a sequence of statements can be treated as one statement by enclosing it in braces {}. We will call this a statement block and denote it in the statement descriptions as {BLOCK}

F.14.1.1 Exit

Format

```
exit
```

Description

The exit statement causes the SMSL program to immediately end and return control to the process that called it. The exit statement must be terminated with a semicolon when used in a SMSL program.

F.14.1.2 Export

Format

export variable

export function function

Parameters

Description

The export statement makes a variable or function in a SMSL library available for export to another SMSL library or program using the requires statement. Each export statement can specify a single variable or function.

Global variables and functions need not be declared before the export statement. The export statement does not require that a variable be explicitly defined within a library, but it does require that it appear in a SMSL statement to create an implicit definition.

Placement of the export statement

The export function function statement can appear before or after the actual function definition. The export variable statement can appear before or after the first appearance of a global variable.

An export statement can appear inside a function definition without any special significance.

Parameter definition

variable name of a SMSL variable that is available for export to another SMSL program

function name of a SMSL function that is available for export to another SMSL program

Errors involving the export statement

The export statement can generate compiler errors in the following instances:

- variable or function is not defined or used in the library;
- variable or function is a SMSL built-in function;
- variable is a local variable of a user-defined function in the library;
- variable or function is duplicated in another export statement;
- variable or function has been imported using the requires statement.

F.14.1.3 Foreach

Format

foreach [list] {BLOCK}

foreach unit variable [list] {BLOCK}

Description

The foreach loop iterates over list and sets variable to be each element of list, performing BLOCK for each element of list in turn.

Parameter definition

list: A list that contains one or more elements that can be equated to variable.

BLOCK: One or more statements that are executed when variable has been equated to an element from list.

unit controls how list is split into individual elements.

Valid Range:

- word assumes that the array elements are separated by white space (spaces, tabs, or \n);
- line assumes that array elements are separated by \n.

default if not specified: line

variable the name of the element that is equated to each element in list.

Examples

The following examples highlight the usage of the foreach statement.

Sum the Elements in an Array

```
sum = 0;

foreach elem ("1\n2\n3\n4\n5")

{

        sum += elem;

}
```

List the Login ID of Each Account on the System

```
foreach user (cat ("/etc/passwd"))

{

        printf(ntharg (item, 1, ":"), "\n");

}
```

NOTE – cat() and ntharg() are built-in SMSL functions.

Count the Number of Words in a String

```
words = 0;

foreach word w ("The cat sat on the mat.")

{

        words++;

}
```

F.14.1.4 Function

Format

```
        function name(argument_list) {BLOCK}
```

Description

The function statement provides user-defined functions within SMSL programs similar to those available in the C-programming language. The function keyword is required in a user function definition. Two additional keywords, local and return, are optional:

- local declares variables that will be used only within the function;
- return identifies function output that is returned to the caller.

Functions must be defined before their first use, and the correct argument_list must be passed in a function call. A function call always returns a character string representing a character string or numeric value. (All data types are represented within SMSL as character strings.)

Parameter definition

name: Character label that is used to identify and call the function from within the SMSL program; name cannot be identical to either of the following:

- a SMSL built-in function;
- a SMSL variable.

argument_list: Zero or more SMSL variables that are passed to the function as parameters when it is called for execution. argument_list can be a NULL entry if no variables are passed to the function, a single argument, or several arguments separated by commas.

BLOCK: One or more SMSL statements that define the action the function performs.

Arguments are passed-by-value to parameters (that is, local copies are created of the arguments' data passed in), and thus changing a parameter will not affect the value of the argument. Function parameters are local to a function and can have names the same as global variables (or the same as parameters of other functions).

If a function definition appears in the middle of executable statements and control flow reaches that definition from above, the definition is skipped as if it were a comment. The only way to enter the body of a function is to explicitly call it. The function definitions serve merely to define a function and are not invoked until called. Hence, it is possible to place executable code above, below, and between function definitions.

F.14.1.5 The return statement

There are three ways to exit a user-defined function:

- return with a return value;
- return without a return value (return value = NULL string);
- fall through to the bottom right brace (return omitted, no return value).

SMSL does not interpret falling through the bottom of a function as an error condition.

SMSL produces a compilation warning similar to that produced by C compilers when it encounters return statements within a function some of which have return values and while others do not. Having multiple exit points in a function that exit in different ways may indicate confusion over whether the function was defined to be perform an action or return a value.

F.14.2 Defining local variables

User-defined function local variables are declared using the local keyword inside the body of the function. The local keyword declares one or more variables specified in a comma-separated list that is terminated by a semicolon. These names become local variables to the function. Following is an example of local variable definitions:

```
function f()

{

        local x;

        local a,b;

        # ... Statements for the function execution.

}
```

Local variables cannot have the same name as a function parameter or another local variable in the same function. Local variable names in one function do not affect those in another function. Local variables can have the same name as a global variable and can "hide" a global name this way.

Local variable declarations are treated as expressions and can appear anywhere within the function that an expression is valid. However, there is no concept of inner scopes in inner blocks and a local variable has scope extending from its point of declaration to the end of the enclosing function (not the enclosing block).

Local variables are initialized to the empty string every time the function is entered. They do not retain their values from a previous call.

The maximum number of local variables and function parameters in user-defined functions (except for the main() function) is implementation specific.

F.14.3 Entry point function

The SMSL entry point function is the main() function. If a SMSL program contains a user-defined function named main, execution begins at the first statement in main(). The SMSL program terminates normally when main() returns. The function you specify as the entry point is permitted to have the same properties as main().

The main() function or the entry point function must be defined in the top-level SMSL program and not in any imported libraries. Functions imported from libraries are ignored when determining whether an entry point function is available.

F.14.3.1 Start of execution without an entry point function

If there is no main() function and no entry point function specified using the SMSL compiler -e option, execution begins at the first executable statement that is not inside a function definition. A program without an entry function will normally have function definitions at the top (they must be defined before their first use) and the main executable statements afterwards. A typical example would be the following:

```
function max(x,y)
{
        if(x > y)
        {
            return x;
        }
        else
        {
            return y;
        }
}
m = max(1,2);              # Execution starts here
printf("maximum is ", m, "\n");
```

As indicated, program execution begins immediately after the function definition.

F.14.3.2 Limitations of user-defined functions

User-defined functions are subject to the following limitations:

- Function Calls are Non-Recursive.
- User-defined functions can make unlimited calls to other functions provided that there is no direct or indirect recursion in the sequence of calls.
- Pass by Value and Pass by Reference Supported.
- SMSL functions support argument passing by reference and argument pass-by value.

Parameter and local variable limits

The parameter and local variable limits for SMSL functions are not defined in this Specification. They are implementation specific.

Function nesting not permitted

SMSL does not permit function nesting – each function definition must be at global scope and cannot be defined inside any other function.

F.14.4 If

Format

```
if (expression) {BLOCK}
if (expression) {BLOCK} else {BLOCK}
if (expression) {BLOCK} elsif (expression) {BLOCK} . . . else{ BLOCK}
```

Description

The if statement is straightforward. Because a statement BLOCK is always bounded by braces, there is no ambiguity about which if, elsif, and else goes with.

Examples

The following examples highlight the usage of if, elsif, and else:

An if statement

```
if (x > 10)
{
        x = 10;          # don’t let x get bigger than 10
}
```

Parameter definition

expression: A SMSL statement whose evaluation returns either TRUE or FALSE.

BLOCK: One or more SMSL statements that are executed once in accordance with the evaluation of the if or elsif expressions.

Description

The if statement is straightforward. Because a statement BLOCK is always bounded by braces, there is no ambiguity about which if, elsif, and else goes with.

Examples

The following examples highlight the usage of if, elsif, and else.

```
An if statement
if (x > 10)
{
       x = 10;     # don’t let x get bigger than 10
}
An if . . . else Statement
if (x == 0)
{
       # do something
}
else
{
       # x != 0
       # do something else
}
An if . . . elsif . . . else Statement
if (x == 0)
{
       # do something
}
elsif (x == 1)
{
```

```
        # do something else
}
else
{
        # x !== 0 &&  x != 1
        # do something else
}
```

F.14.5 Last

Format

last

Description

The last statement causes SMSL execution to exit the innermost execution loop. The last statement must be terminated with a semicolon when used in a SMSL program.

F.14.6 Next

Format

next

Description

The next statement immediately starts the next iteration of the innermost execution loop.

F.14.7 Requires

Format

requires library

Description

The requires statement imports variables and functions identified in export statements from a previously created SMSL library into the SMSL program. Each requires statement can specify a single library name.

SMSL contains no explicit import statement; using the requires statement implies importation. The requires statement searches for the binary containing the library and reads all its export statement information, importing the specified variables and/or functions into the SMSL program.

Any number of requires statements can appear in a SMSL program. All libraries specified in requires statements must be available to the compiler during compilation.

Requires statements in imported libraries

The SMSL compiler will automatically resolve nested dependencies in imported libraries, but it will not automatically load all the other exported functions and variables found in the library that satisfies the nested dependency. You must explicitly import a library in order to guarantee access to all the exported variables and functions within it.

Parameter definition

library: Name of the library whose specified export variables and functions are to be imported into the SMSL program.

A requires statement can appear inside a function definition without special significance.

Variable and function availability among imported libraries

When a SMSL program imports variables and functions from more than one library, the imported variables and functions from one library can set and use the imported variables and functions from the others, regardless of how the libraries are loaded for compilation.

Errors involving the requires statement

The requires statement can generate compiler errors in the following instances:

- A reference to an imported variable or function appears before the requires statement that imports it. You must place a requires statement before the first use of the imported variable or function.
- An imported function has the same name as a function defined within the SMSL program.
- The same variable or function name is imported from two or more libraries.

F.14.8 Switch

Format

```
switch (variable)
{
    case a: {BLOCK}
    case b: {BLOCK}
    . . .
    case p,q,r:{ BLOCK}
    . . .
    case n: {BLOCK}
    default: {BLOCK}
}
```

Parameters

Description

The switch statement evaluates variable and based on its integer value executes a specific SMSL BLOCK. The case labels correspond to the values of variable for which a specific SMSL BLOCK is available.

If the value of variable falls outside the range of the values in the case labels, execution continues with the BLOCK corresponding to the default label. If no default label exists, execution will continue with the first statement following the switch statement.

Parameter definition

variable SMSL: Variable name whose integer value specifies the SMSL statement BLOCK that will be executed.

a,b, . . . p,q,r, . . . n Integer values indicating the value of variable that will cause the corresponding BLOCK to be executed.

BLOCK: One or more statements that are executed when the corresponding case value equals variable.

Description

The switch statement evaluates variable and based on its integer value executes a specific SMSL BLOCK. The case labels correspond to the values of variable for which a specific SMSL BLOCK is available.

If the value of variable falls outside the range of the values in the case labels, execution continues with the BLOCK corresponding to the default label. If no default label exists, execution will continue with the first statement following the switch statement.

The SMSL switch statement behaves the same way as a long sequence of if-then-else-if statements. A case or default clause is effectively a run-time statement that specifies a comparison against the value of variable:

- If the value of variable matches a case, execution moves inside the BLOCK for the case or default clause; and after completing BLOCK, execution continues after the entire switch statement (that is, there is no falling through to the next case clause).
- If the value of variable does not match a case, execution skips to the default clause; and if there is none, execution moves to the statement following the switch statement.

Any statement within the switch statement case block that is not part of a case or default BLOCK executes only if all the case labels above it failed to match variable (that is, it executes as part of the normal sequence of control flow).

The following are the properties of the SMSL switch statement:

- Case expressions can be dynamically evaluated expressions and constant expressions.
- The colon delimiter that separates the case label from the executable BLOCK is optional in SMSL.
- SMSL requires that the default label follow all case labels in the switch statement case block. It returns a compilation error if one or more case labels follow default.
- SMSL does not return a compilation error for duplicate case labels in the switch statement. In SMSL, the second of the duplicate case labels is unreachable.
- SMSL allows multiple cases that execute a common BLOCK to be specified as a comma separated list within a single case label. (Conversely, the stacked labels will not work in SMSL.)
- Execution of a SMSL BLOCK does not “fall through” to the next case label and BLOCK. Upon reaching the closing right brace of a case or default BLOCK, execution moves to the end of the SMSL switch statement.
- The SMSL switch statement uses the last statement to exit from a BLOCK. The last statement exits the innermost switch statement or loop. However, because of the absence of “fall-through” in SMSL, there is little need to use the last statement in the switch statement.
- SMSL generates a compiler error upon detecting two default labels in a single switch statement.
- SMSL permits nested switch statements.

The case BLOCKs are evaluated at run-time in their order of appearance:

- case order for BLOCKs;
- left-to-right for expressions in the comma-separated lists of multiple-case labels.

All expressions within a comma-separated list are evaluated before the case label. This evaluation occurs even if the first expression is a match.

This sequence and method of evaluating the case label can be a dangerous pitfall if any expression in the list modifies either variable for the current switch statement or a variable used in another case expression. Under SMSL, statements within a switch statement that are not part of a BLOCK (free statements) can and will be executed if they are reached by the flow of execution. The condition for control flow to reach these statements is that variable cannot match any of the case labels that precede them within the switch statement. SMSL does not return a warning or error message when two case labels evaluated against variable are nested one inside the other. Two examples of this situation are shown in the following SMSL switch example:

switch(x) { case 1: { f1() # Function f1 Called case 2 : {f2();} # Function f2 Unreachable f3(); # Function f3 Called } default: {case 4: {f4();}} # Function f4 called if x=4 }

Since case and default labels are run-time statements, the effect of one case label nested within another is that variable must match the case value for the case BLOCK to execute. This means that variable must equal two different values! In case 1 of the example, f2 will never be called because x cannot equal both 1 and 2.

In the default case of the example, f4 will be called if variable = 4 because there is no case 4 defined in the switch statement. When variable = 4, the default BLOCK executes, containing the case 4 BLOCK call to function f4.

F.14.9 While

Format

while (expression) {BLOCK}

Parameters

Description

The while loop executes statements as long as expression evaluates to TRUE (non-zero).

Example

The following sample SMSL statements print the integers from 1 to 10.

```
x = 1;

while (x <= 10)

{

        printf (x, " ");

        x++;

}

printf ("\n");
```

Parameter definition

expression A SMSL statement whose evaluation returns either TRUE or FALSE.

BLOCK: One or more SMSL statements that execute repeatedly as long as expression evaluates to TRUE.

Description

The while loop executes statements as long as expression evaluates to TRUE (non-zero).

Example

The following sample SMSL statements print the integers from 1 to 10.

```
x = 1;

while (x <= 10)

{

        printf (x, " ");

        x++;

}

printf("\n");
```

F.14.10 Object model

SMSL is based on a simple object-oriented model which makes it possible to do a mapping from systems management environment described in GDMO (see CCITT Rec. X.733 | ISO/IEC 10164-4). An object is a construct with properties that are variables or other object. Functions associated with an object are the object's methods.

A SMSL user can access the properties of an object with the following notation:

objectName.propertyName

This object can be either GDMO object or some locally defined object.

If this object is GDMO object, the mapping from GDMO properties to SMSL properties is shown in Table F.13.

Table F.13 – Mapping between GDMO and SMSL

GDMO property	SMSL property
class-label	name of object type
instance identifier(INTEGER, etc.)	value of object instance variable
initial values in creating an instance	parameters of “new” operation
managed object instance names	ASN.1 value notation
ASN.1 type of ATTRIBUTES	name of variables
label of ATTRIBUTE GROUPS	name of array variable
label of ACTION	name of method (function)
asynchronous NOTIFICATION handling	onEvent(discriminatorConstruct) handler

A property can be defined by assigning it a value as follows:

objectName.propertyName = value;

A method is a function associated with an object. A function can be associated with an object as follows:

objectName.methodName = functionName

where object is an existing local object, method is the name assigned to the method, and functionName is the name of the function.

Method in the context of the object can be called as follows:

objectName.methodName(parameters);

F.14.10.1 GDMO operation and action parameters

Operation and action parameters are passed to the action method as an object mapped by the rules described in the object model. Return parameter values are returned from the action method (function) as an object mapped by these rules. The description format is as follows:

outputObjectName = ObjectName.actionName(inputObjectName);

outputObjectName = ObjectName.operationName(inputObjectName);

F.14.10.2 “this” object reference

SMSL has a special keyword, “this”, that can be used to refer to the current object.

this[propertyName]

F.14.10.3 Creating and deleting objects

new

An operator that lets you create an instance of a user-defined object type.

Creating an object type requires two steps:

1) Define the object type by writing a function.

2) Create an instance of the object with new.

To define an object type, create a function for the object type that specifies its name, properties and methods. An object can have a property that is itself another object.

Format

objectName = new objectType (param1 [,param2] .... [,paramN])

objectName is the name of the new object instance.

objectType is function that defines an object type.

param1..paramN are the property values for the object. These properties are parameters defined for the objectType function.

An instance of a class can be deleted with “delete” operator.

delete objectName

objectName is the name of the existing object instance.

Moreover local object type can be defined by writing a function. Then a local instance of the object can be created with “new” operator.

Object expression to represent name-binding

If two objects have a name-binding relationship, the following expression of that subordinate object is allowed:

superiorObjectInstance.subordinateObjectInstance

F.14.10.4 WITH

A statement that establishes a default object for a set of statements. Within the set of statements, any property references that do not specify an object are assumed to be for the default object.

Syntax

with (objectName){

statements

}

objectName specifies the default object to use for the *statements*. The parentheses are required around *objectName*. *statements* is any block of statements.

F.14.10.5 Event handler

onEvent

Description

An event handling operator to describe the processing when a specified notification defined by GDMO occurs.

Syntax

onEvent(*DiscriminatorConstructValue*){

statements

}

DiscriminatorConstructValue is DiscriminatorConstruct type value. *statements* is any block of statements.

F.14.10.6 triggerParameterCount

This SMSL virtual machine keeps track of the number of trigger parameters by means of this variable.

F.14.10.7 triggerArgument

Description

The triggerArgument accepts the list of trigger parameters which can be accessed from an SMSL script and returns the identity of the launch pad which then executes the script. The argument list includes the trigger id and script id unless the trigger is not parameterized.

Syntax

triggerArgument(*argument_list*), where *argument_list* can have up to *triggerParameterCount* elements.

Example of SMSL script for management of EFD

This example describes how an EFD can be created and its operational state determined.

The parameters to the script to be triggered are the script id and the notification destination. The SMSL script to get the operational state attribute of the EFD and return it as a notification to the destination is given below.

The trigger is specified with the following parameters:

- *triggerId*, the identifier of this trigger;
- *scriptId*, the identifier of the script to be executed;
- *managerId*, the identifier of the destination to which event reports are to be forwarded.

The SMSL script for this example is given below:

```
#include “CMIP.CMIP-1.h”
#include “DMI.Attribute-ASN1Module.h”

manager1 = triggerArgument(triggerId, scriptId, managerId);

/* Instance creation as CMISFilter type */
CMISFilter counterValue _GT10 = item ( greaterOrEqual ( Attribute-ASN1Module.Count, 10))

/*
* Creating instance of EFD with the following parameters.
* DiscrimnatorConstruct = ( counter > 10 ),
* administrativeState = default value, destination = manager1
*/

efd1 = new eventForwardingDiscriminator( counterValue_GT10, manager1);

triggerResult = efd1.operationalState;                    /* get operational state */

printf(“operational state of event forwarding discriminator %d \n”, triggerResult);
```

The destination manager is supplied as a trigger parameter. The trigger causes the launch pad to spawn threads in order to execute all the script instructions in sequence. The launch pad passes appropriate parameters to threads. When creating efd1, for example, the launch pad passes the script id and the destination manager as parameters to the thread.

F.14.11 BNF for SMSL

(Conventions: "{}" is used for productions which may occur 0 or more times; "[]" is used for optional productions.)

***Tokens**:*

```
T_ELLIPSIS              : "..."
T_EQ                    : "=="
T_NE                    : "!="
T_REGEXP_EQ             : "=~"
T_REGEXP_NE             : "!~"
T_LEQ                   : "<="
T_GEQ                   : ">="
T_GT                    : ">"
T_LT                    : "<"
T_AND                   : "&&"
T_OR                    : "||"
T_NOT                   : "!"
```

```
T_INC                   : "++"
T_DEC                   : "--"
T_PLUSEQ                : "+="
T_MINUSEQ               : "-="
T_MULEQ                 : "*="
T_DIVEQ                 : "/="
T_MODEQ                 : "%="
T_BITANDEQ              : "&="
T_BITOREQ               : "|="
T_LEFT_SHIFT            : "<<"
T_RIGHT_SHIFT           : ">>"
T_RIGHT_SHIFT_ASSIGN    : ">>="
T_LEFT_SHIFT_ASSIGN     : "<<="
T_XOR_ASSIGN            : "^="
```

Keywords :

```
T_IF                    : "if"|"IF"
T_ELSE                  : "else"|"ELSE"
T_ELSIF                 : "elsif"|"ELSEIF"
T_FOREACH               : "foreach"|"FOREACH"
T_FOR                   : "for"|"FOR"
T_WORD                  : "word"|"WORD"
T_LINE                  : "line"|"LINE"
T_NEXT                  : "next"|"NEXT"
T_LAST                  : "last"|"LAST"
T_WHILE                 : "while"|"WHILE"
T_DO                    : "do"|"DO"
T_UNTIL                 : "until"|"UNTIL"
T_EXIT                  : "exit"|"EXIT"
T_TRUE                  :
"true"|"TRUE"|"True"|"yes"|"YES"|"Yes"
T_FALSE                 :
"false"|"FALSE"|"False"|"no"|"NO"|"No"
T_RETURN                : "return"|"RETURN"
T_LOCAL                 : "local"|"LOCAL"
T_FUNCTION              : "function"|"FUNCTION"
T_NATIVE                : "native"|"NATIVE"
T_RPC                   : "rpc"|"RPC"
T_VOID                  : "void"
T_REQUIRES              : "requires"|"REQUIRES"
T_EXPORT                : "export"|"EXPORT"
```

```
T_CASE                  : "case"|"CASE"
T_SWITCH                : "switch"|"SWITCH"
T_DEFAULT               : "default"|"DEFAULT"
```

Rules:

```
program : stmts

stmts : { stmt }

stmt : expr ';'          |
       return    |
       if    |
       foreach    |
       switch     |
       case      |
       default    |
       while     |
       do_until  |
       for_loop  |
       T_LAST ';' |
       T_NEXT ';' |
       T_EXIT ';' |
       function   |
       native_function  |
       rpc       |
       local_var_dec    |
       export    |
       requires

requires : T_REQUIRES requires_name ';'

library_name : T_STRING | T_IDENTIFIER

requires_name : library_name

export : T_EXPORT [ T_FUNCTION ] export_name ';'

export_name : T_IDENTIFIER

part : T_WORD
```

```
foreach : T_FOREACH part simple_id '(' expr ')' '{' stmts '}'

case_exprs : { case_expr ',' } case_expr

case_expr : expr

case : T_CASE case_exprs optional_colon '{' stmts '}'

default : T_DEFAULT optional_colon '{' stmts '}'

optional_colon  :  [ ':' ]

switch : T_SWITCH '(' expr ')' '{' stmts '}'

for_loop : T_FOR '(' optional_expr ';' optional_expr ';' optional_expr ')'
          '{' stmts '}'

do_until : T_DO '{' stmts '}' T_UNTIL '(' expr ')' ';'

while : T_WHILE '(' expr ')' '{' stmts '}'

void : [ T_VOID ]

native_function :
       T_NATIVE void T_FUNCTION function_name '(' func_param_list ')' ';'

rpc : T_RPC void T_FUNCTION function_name '(' func_param_list ')' ';'

function : T_FUNCTION function_name '(' func_param_list ')' '{' stmts '}'

func_param_list : param_list

function_name : T_IDENTIFIER

param_list : [ { one_param ',' }  one_param ]

one_param : T_IDENTIFIER |
           T_ELLIPSIS

local_var_dec : T_LOCAL var_list ';'
```

```
var_list : { one_var ',' }  one_var

one_var : T_IDENTIFIER

return : T_RETURN [ expr ] ';'

if : T_IF '(' expr ')' '{' stmts '}' opt_elsifs opt_else

opt_elsifs : { elsif }

elsif : T_ELSIF '(' expr ')' '{' stmts '}'

opt_else : [ else ]

else : T_ELSE '{' stmts '}'

optional_expr : [ expr ]

expr : unary_expr             |
       expr '+' expr               |
       expr '-' expr               |
       expr '*' expr               |
       expr '/' expr               |
       expr '%' expr               |
       expr T_EQ expr              |
       expr T_NE expr              |
       expr T_REGEXP_NE expr       |
       expr T_REGEXP_EQ expr       |
       expr T_LT expr              |
       expr T_GT expr              |
       expr T_LEQ expr             |
       expr T_GEQ expr             |
       expr T_AND expr             |
       ternary_expr           |
       expr T_OR expr              |
       expr '|' expr               |
       expr '&' expr               |
       expr '^' expr               |
       expr T_LEFT_SHIFT expr      |
       expr T_RIGHT_SHIFT expr           |
       lvalue '=' expr             |
```

```
        lvalue T_PLUSEQ expr          |
        lvalue T_MINUSEQ expr         |
        lvalue T_MULEQ expr           |
        lvalue T_DIVEQ expr           |
        lvalue T_BITANDEQ expr        |
        lvalue T_BITOREQ expr         |
        lvalue T_MODEQ expr           |
        lvalue T_LEFT_SHIFT_ASSIGN expr    |
        lvalue T_XOR_ASSIGN expr           |
        lvalue T_RIGHT_SHIFT_ASSIGN expr   |
        expr '.' expr

simple_id : T_IDENTIFIER

ternary_expr : expr '?' expr ':' expr

lvalue : simple_id

unary_expr : primary          |
             '-' unary_expr   |
             T_NOT unary_expr |
             T_INC lvalue     |
             T_DEC lvalue

function_call_id : T_IDENTIFIER

primary : simple_id                      |
          T_INT                  |
          T_FLOAT                      |
          T_STRING                     |
          T_TRUE                 |
          T_FALSE                      |
          '(' expr ')'                 |
          lvalue T_INC                 |
          lvalue T_DEC                 |
          function_call_id '(' arglist ')'

arglist : [ { expr ',' } expr ]
=======================================================
```

Note that the definition of string should allow embedded \" within strings.

```
T_IDENTIFIER            : [A-Za-z_][A-Za-z_0-9]*
T_STRING                : \"[^"]*\"
T_FLOAT                 : [0-9]*"."[0-9]+
T_INT                   : [0-9]+
```

/* Operator tokens and their precedences */

```
%right '=' T_PLUSEQ T_MINUSEQ T_MULEQ T_DIVEQ T_MODEQ T_BITANDEQ T_BITOREQ
T_LEFT_SHIFT_ASSIGN T_RIGHT_SHIFT_ASSIGN T_XOR_ASSIGN
%left '?' ':'
%left T_OR
%left T_AND
%left '|'
%left '^'
%left '&'
%left T_EQ T_NE T_REGEXP_EQ T_REGEXP_NE
%left T_LT T_GT T_LEQ T_GEQ
%left T_LEFT_SHIFT T_RIGHT_SHIFT
%left '+' '-'
%left '*' '/' '%'
%left '.'
%right T_UNARY T_NOT T_INC T_DEC
%left '('
%left '['
```

Annex G

SMSL support functions

(This annex forms an integral part of this Recommendation | International Standard)

acos()

Return the arccosine of the argument.

Format

acos(*cosine*)

Parameter

Parameter	Definition
cosine	cosine argument valid range: –1 <= *cosine* <= 1

Description

The acos() function returns the arccosine of *cosine*; that is, the length in radians of the arc whose cosine is *cosine*.

The output range for the acos() function is $0 \leq \text{acos}() \leq p$. The acos() function return value is accurate to six decimal places.

asctime()

Return the date and time as a character string.

Format

asctime(*clock,format*)

Parameters

Parameter	Definition
clock	A reference to the clock or timer whose value should be converted to a character string. *clock* is most commonly time().
format	Optional format specification for the asctime() output string. The following field specifiers are valid: %a abbreviated weekday %A full weekday %b abbreviated month %B full month %c local date and time representation %d decimal day of the month (from 01 to 31) %E combined Emperor/Era name and year %H decimal hour in 24-hour mode (from 00 to 23) %I decimal hour in 12-hour mode (from 01 to 12) %j decimal day of the year (from 001 to 366) %m decimal month (from 01 to 12) %M decimal minute (from 00 to 59) %n new-line character %N Emperor/Era name %o Emperor/Era year %p equivalent of AM/PM

Parameters
(concluded)

Parameter	Definition
format	Optional format specification for the asctime() output string. The following field specifiers are valid: %S decimal second (from 00 to 61) %t tab character %U decimal week of the year: Sunday is the first day of the week; all days preceding the first Sunday of the year are in week 0 (from 00 to 53) %w decimal day of the week: Sunday is the first day of the week (from 00 to 06) %W decimal week of the year: Monday is the first day of the week; all days preceding the first Monday of the year are in week 0 (from 00 to 53) %x local date representation %X local time representation %y decimal year without century (from 00 to 99) %Y decimal year with century %Z time zone name (if time zone name exists) %% % character Field specifiers may be expressed as: [– | 0] *field_specifier*.p where – left-justify the field (right-justification is the default) 0 right-justify the field and pad with zeros on the left .p minimum number of digits to display for decimal fields or the maximum number of characters to display for alphabetic fields. For decimal fields, empty character positions are filled with leading zeros. For character fields, excess characters are truncated on the right. Default if not specified: 24-character string with the format: Sun Sep 16 01:03:52 1973

Description:

The asctime() function returns the date/time of clock as a character string. It is equivalent to the C- library asctime() function.

If format is given, asctime() returns the date/time string in the specified format. The field specifiers used in format are equivalent to those used in the C-library strftime() function.

asin()

Return the arcsine of the argument.

Format

asin(*sine*)

Parameter

Parameter	Definition
sine	Valid range: $-1 \leq sine \leq 1$

Description

The asin() function returns the arcsine of *sine*; that is, the length in radians of the arc whose sine is *sine*. The output range for the asin() function is $-\pi/2 \leq \text{asin}() \leq \pi/2$.

atan()

Return the arctangent of the argument.

Format

atan(*tangent*)

Parameter

Parameter	Definition
tangent	Valid range: $-\infty \leq tangent \leq \infty$

Description

The atan() function returns the arctangent of *tangent*; that is, the length in radians of the arc whose tangent is *tangent*.

The output range for the atan() function is $-\pi/2 \leq \text{atan}() \leq \pi/2$.

cat()

Return the content of a file as a single text string.

Format

cat(*filename*)

Parameters

Parameter	Definition
filename	Name of the file whose contents are to be returned

Description

The cat() function returns the contents of file *filename* as a single string or the NULL string on error. New-lines are preserved so that the foreach statement can be used to process the returned string as a list of the lines in *filename*.

Example

The following SMSL statements list the names of users listed in the UNIX system password file.

```
people = cat("/etc/passwd");
foreach person (people)
{
name = ntharg(person, 1, ":");
printf("name of person is:%s", name, "\n");
}
```

ceil()

Return the smallest integer that is not less than the argument.

Format

ceil(*argument*)

Parameter

Parameter	Definition
argument	Numeric argument whose least integer upper bound is to be determined

Description

The ceil() function returns the smallest integer that is not less than *argument*; that is, the least integer upper bound for *argument*.

The ceil() function and the floor() function together bracket *argument* such that the following are true:

- If *argument* is an integer: ceil(*argument*) = *argument* = floor(*argument*).
- If *argument* is not an integer: floor(*argument*) < *argument* < ceil(*argument*) and ceil(*argument*) = floor(*argument*) + 1.

chan_exists()

Verify that a process or file channel exists.

Format

chan_exists(*channel*)

Parameter

Parameter	Definition
channel	The process or file I/O channel name (shared channels) or number (local channels) that is being verified

Description

The chan_exists() function returns 1 if the local or shared channel exists and 0 if it does not.

The chan_exists() function return value can be used with condition variables for synchronizing one SMSL process to wait until another has opened a channel using either the popen() or fopen() function.

close()

Close a file or process channel.

Format

close(*channel*,*flags*)

Parameters

Parameter	Definition
channel	The process or file I/O channel name (shared channels) or number (local channels) that is to be closed
flags	Optional bit flags used to control close execution. The following bits are used: Bit 1 = 1 indicates that any system process associated with the channel (that is, the SMSL fopen() or popen() function) should be killed while closing the channel. Bit 2 = 1 indicates that the channel should be closed even if another SMSL process is blocked waiting for a read(), readln(), or write() function. Bit 2 applies only to global channels and is ignored by local channels. Default if not specified: Bits 1 and 2 are both zero.

Description

The close() function closes a channel to a process or command previously created by a fopen() or popen() call.

When flags is not specified, the default is zero.

When bit 1 = 0, the close() function does not kill any processes spawned as a result of the fopen() or popen(); and these processes are allowed to continue. This feature of close() allows you to open a channel to a SMSL process, send additional data, and close the channel while allowing the process to complete.

When bit 2 = 1, the close() function will close the channel even if another SMSL process is blocked pending an I/O request on that channel. When blocking occurs, close() causes the blocked function to wake and receive an error return and errno from the process to which the channel was opened.

The close() function returns the NULL string if the closure was successful and –1 with the SMSL variable errno set if the closure was unsuccessful. The close() function fails when bit 2 = 0 and channel is a global channel with at least one blocked SMSL process.

Example

```
# close the channel represented by variable chan

close(chan);
```

concat()

Concatenate two strings.

Format

concat(*string1*, *string2*)

Description

The concat function causes the concatenation of two strings.

For example, concat(“ab”,”cd”) returns “abcd”.

cond_signal()

Signal a process that is blocked on a condition wait.

Format

cond_signal(*condition_variable,all*)

Parameters

Parameter	Definition
condition_variable	Name of the variable that will unblock a process blocked by the cond_wait() function
all	Non-NULL value that directs the cond_signal() function to unblock all SMSL processes that are blocked waiting for condition_variable

Description

The cond_signal() function can signal another SMSL process that is currently blocked for a cond_wait() function on *condition_variable*. If *all* is specified and is not the NULL string, the cond_signal() function will wake all SMSL processes that are blocked on condition_variable. If no processes are blocked on condition_variable, the cond_signal() function has no effect. The cond_signal() function can never block and always returns the NULL string.

cond_wait()

Block a process until a condition signal is received.

Format

cond_wait(*condition_variable,lockname,timeout*)

Parameters

Parameter	Definition
condition_variable	Name of the variable that will end the cond_wait() condition. *condition_variable* is issued by the cond_signal() function.
lockname	The name of the lock the cond_wait() function should attempt to acquire when it receives the correct unblocking *condition_variable* in the cond_signal() function. If *lockname* is the NULL string, the cond_wait() function will not attempt to acquire a lock after receiving *condition_variable*.
timeout	Number of seconds to wait for the receipt of *condition_variable* before unblocking and releasing *lockname*. Valid range: *timeout* > 0 specifies the timeout value in seconds; *timeout* < 0 specifies an infinite timeout; only the receipt of *condition_variable* can unblock the process; *timeout* = 0 is not permitted and will result in a SMSL run-time error message. Default if not specified: Infinite timeout.

Description

The cond_wait() function blocks the current SMSL process until either *condition_variable* is received or until timeout expires. If the SMSL process holds *lockname* when the cond_wait() function is issued, the cond_wait() function releases the lock. When the cond_wait() function receives condition_variable, the cond_wait() function immediately attempts to acquire *lockname*.

If the cond_wait() function returns a 1, it will always hold an exclusive lock on *lockname*. If the cond_wait() function fails, it will not hold any form of lock on *lockname* when it returns. If a timeout occurs, the cond_wait() function returns a failure value of "0," sets the SMSL errno to E_SMSL_TIMEOUT and will not hold any lock on *lockname*.

condition_variable

condition_variable is the name of the condition variable that the cond_wait() function waits to have signaled by the cond_signal() function. Condition variable names have global scope analogous to locks and shared channels.

None of these different global scopes interfere with one another. You can use the same name without conflict for a lock, a shared channel, and a condition variable.

lockname

On entry to the cond_wait() function, the process releases the lock *lockname* and blocks waiting to be signalled. *lockname* should usually be an exclusive lock held by this process; otherwise, run-time error messages may occur (although the cond_wait() function will still try to go ahead and wait for a signal anyway). The cond_wait() function will always block waiting for the cond_signal() function or for timeout.

When another SMSL process performs a cond_signal() function that wakes this SMSL process, the cond_wait() function call will attempt to gain an exclusive lock (if a lock is requested; that is, if *lockname* is not the empty string) and either return immediately with the lock or join the queue waiting for an exclusive lock on *lockname*.

It is common style to supply *lockname* since condition variables are almost always shielded by locks. In the cond_wait() function, *lockname* must be supplied as the NULL string rather than omitted to force the SMSL coder to consider whether a lock is needed. The required *lockname* will reduce the number of errors caused by not using a lock when one is needed.

timeout

timeout behaviour is unchanged regardless of whether the cond_wait() function is waiting for a cond_signal() function or waiting to acquire *lockname*. If *condition_variable* or *lockname* is destroyed before the cond_wait() function is complete, the cond_wait() function returns 0 and sets the SMSL errno value but will not hold any lock on *lockname*.

lockname can be the NULL string, in which case *condition_variable* is considered to have no associated lock; and the cond_wait() function will return success immediately upon being signaled without waiting for any lock.

cos()

Return the cosine of the argument.

Format

cos(*radians*)

Parameter

Parameter	Definition
radians	Arc length in radians whose cosine is to be determined Valid range: $-\infty \leq$ radians $\leq \infty$

Description

The cos() function returns the cosine of radians.

The output range for the cos() function is $-1 \leq \cos() \leq 1$.

cosh()

Return the hyperbolic cosine of the argument.

Format

cosh(*argument*)

Parameter

Parameter	Definition
argument	Numeric value whose hyperbolic cosine is to be determined Valid range: $-\infty \leq argument \leq \infty$

Description

The cosh() function returns the hyperbolic cosine of argument. The hyperbolic cosine is defined by the expression:

$$\cosh(x) = (e\ x + e\ -x\ )/2$$

where e is the base for the natural logarithms (e = 2.71828 . . .). The output range for the cosh() function is $1 \leq \cosh() \leq \infty$.

date()

Return the date and time as a 24-character string.

Format

date()

Description

The date() function returns the current date and time as a 24-character string in the format:

Sun Sep 16 01:03:52 1973

The date() function is equivalent to the C-library ctime(3) function. The date() function is also equivalent to the SMSL statement:

asctime(time());

Example

The following examples highlight the usage of the date() function.

Assign the Current Date and Time to a Variable:

today = date();

debugger()

Suspend process pending an attach command from the SMSL debugger.

Format

debugger()

Description

The SMSL debugger() function suspends the current SMSL process waiting for an attach command from the SMSL debugger. The debugger() function complements options within the SMSL debugger that suspend SMSL processes. The debugger() function offers a low-level method of stopping a SMSL process for debugging before it begins.

Although modifying SMSL source code to debug a particular script may not be convenient, the debugger() function provides a general method whereby all SMSL code can be debugged.

The only way to restart a SMSL function suspended through the debugger() function is through the SMSL debugger. If the SMSL process is already being processed by the SMSL debugger, a call to the debugger() function call has no effect. The debugger() function always returns the NULL string.

destroy()

Destroy a SMSL object.

Format

destroy(*object, description*)

Parameters

Parameter	Definition
object	The alphanumeric identifier for the object. *object* is assigned when the object is created.
description	Optional text string that can be used to explain why the object was destroyed. The text string must be enclosed in double quotation marks.

Description

The destroy() function deletes the application instance object. The destroy() function returns TRUE on success, and FALSE on error.

Example

destroy object whose name is in variable <name>

destroy(name);

Default if not specified: NULL string

difference()

Return the list of elements that are unique to a specified SMSL list.

Format

difference(*list1,list2,list3,list4 . . . ,listn*)

Parameters

Parameter	Definition
list1	SMSL list whose elements are being compared against the elements of all other specified lists
list2 . . . listn	One or optionally more lists whose elements are compared against *list1*

Description

The difference() function returns a SMSL list with all elements of *list1* that are not in any of the lists *list2 . . . listn*. If *list1* is the NULL list, the result is the NULL list.

list1 may contain duplicates. Duplicates in list1 appear in the return list in same order and number as they appeared in *list1*, provided that they were not removed by matches with the other lists in the difference() function.

All elements that are returned from *list1* remain in the same order in the return list. If the return list is not the NULL set, the returned set is delimited by new-line characters; that is, all set elements end with a new-line character.

execute()

Execute a command of a specified type.

Format

execute(*type,command,instance*)

Parameters

Parameter	Definition
type	Command processor that should interpret and execute command Valid range: The built-in command types OS or SMSL or a valid user-defined command type.
command	Syntax of the submitted command
instance	The application instance against which command should execute Default if not specified: The application instance that is the nearest ancestor of command.

Description

The execute() function executes a command of any type and returns any output that it produces to stdout or stderr. The status of command is saved in the SMSL variable exit_status.

Example

```
# SQL data is returned into the buffer "data"
data = execute("SQL", "select * from user_objects");
```

exists()

Verify the existence of a SMSL object.

Format

exists(*object,inherit*)

Parameters

Parameter	Definition
object	The alphanumeric identifier for the object whose existence is being verified. *Object* is assigned when the object is created.
inherit	Boolean expression controls whether exists will search the entire inheritance hierarchy to verify the existence of *object*: If *inherit* = TRUE, do not search the inheritance hierarchy. If *inherit* = FALSE and if object is not a reference to an absolute object, search the inheritance hierarchy.

Description

The exists() function returns TRUE if object exists; FALSE otherwise. The exists() function is useful in application discovery procedures that determine whether a discovered instance has previously been discovered and instantiated in the object hierarchy.

Example

```
# Check if we have created the user before
if (exists(name))
{
        printf(“%f”,name);
}
else
{
        printf("User name does not exist");
}
```

exp()

Return the base of the natural logarithms e raised to a power.

Format

exp(*exponent*)

Parameter

Parameter	Definition
exponent	Numeric value to which the natural base e is raised

Description

The exp() function returns the value $e^{exponent}$ where *e* is the base of the natural logarithms (*e* = 2.71828 . . .).

fabs()

Return the absolute value of an argument.

Format

fabs(*argument*)

Parameter

Parameter	Definition
argument	Floating point value whose absolute value is to be determined

Description

The fabs() function returns the absolute value of argument; that is:

- *argument if argument* ≥ 0;
- *–argument if argument* < 0.

file()

Return file information.

Format

file(*filename*,*dummy*)

Parameters

Parameter	Definition
filename	Name of the file whose last modification date is to be returned
dummy	Dummy variable that specifies expanded file information in the form: modtime atime ctime mode size numlinks type modtime is the last modification date expressed as the number of seconds since midnight, January 1, 1970. atime is the last access time expressed as the number of seconds since midnight, January 1, 1970. ctime is the last change of status expressed as the number of seconds since midnight, January 1, 1970. mode is the file permissions expressed as an octal integer. size is the length of the file expressed as a number of characters. numlinks the number of links to the file within the file system. type is a character string indicating the file type: FILE ordinary user data file DIR directory SPECIAL character special file BLOCK block special file FIFO pipe or FIFO LINK symbolic link SOCKET socket (not available on all platforms) UNKNOWN unknown file type, possibly a LINK or SOCKET on platforms where the UNIX stat() function cannot determine the type; that is, where S_ISLINK or S_ISSOCK are undefined.

Description

The file() function returns the last modification time of file filename as the number of seconds since midnight, January 1, 1970. If the file does not exist, the file() function returns the NULL string. This function is useful for testing the existence of a file.

NOTE 1 – The file() function return values depend on the operating system and in some cases the file system. Some non-UNIX platforms may not return all return values or may return one or more meaningless return values. For a specific platform, the file() function will generally return the same information as the C-programming language stat() function.

The user does not need permission to read the file but does require search permission on each directory in the path name leading to *filename*. If the user does not have such permission, the file() function fails and returns the NULL string.

The value of *dummy* is ignored, but its presence causes the file() function to return a more detailed string of information.

Examples

The following examples highlight the usage of the file() function.

Print Last Modification Date of the UNIX System Password File

```
printf(“%s”,asctime(file("/etc/passwd")));
# modification time
```

Test for the Existence of a File

```
if (file("some_file"))
{
    printf("File exits!");
}
else
{
    printf("File does not exist.");
}
```

floor()

Return the largest integer that is not larger than the argument.

Format

floor(*argument*)

Parameter

Parameter	Definition
argument	Numeric argument whose greatest integer lower bound is to be determined

Description

The floor() function returns the largest integer that is not greater than *argument*; that is, the greatest integer lower bound for *argument*.

The floor() function and the ceil() function together bracket *argument* such that the following are true:

If *argument* is an integer: ceil(*argument*) = *argument* = floor(*argument*)

If *argument* is not an integer: floor(*argument*) < *argument* < ceil(*argument*) and ceil(*argument*) = floor(*argument*) + 1

fmod()

Return the floating point remainder of a division operation.

Format

fmod(*dividend*,*divisor*)

Parameters

Parameter	Definition
dividend	The floating point value whose remainder will be returned after being divided by *divisor*
divisor	The floating point value that will divide *dividend*

Description

The fmod() function returns the floating point remainder of the division (*dividend*)/(*divisor*).

fopen()

Open a SMSL channel to a file.

Format

fopen(*filename*,*mode*)

Parameters

Parameter	Definition
filename	Name of the file to which the SMSL channel should be opened
mode	The file access mode. Valid ranges: r Open for read w Truncate to zero length for write or create file for write a Open for append to end of file or create for write rb Open binary file for read wb Truncate binary file to zero length for write or create binary file for write ab Open binary file for append at end of file or create binary file for write r+ Open for read and write (*update*) w+ Truncate to zero length for read and write or create for read and write a+ Open for read and write at end of file or create file for read and write r+b Open binary file for read and write (*update*) w+b Truncate binary file to zero length for read and write or create binary file for read and write a+b Open binary file for read and write at end of file or create binary file for read and write

Description

The fopen() function opens a channel to filename that provides the access to filename from within a SMSL process. The read(), write(), get_chan_info(), share(), and close() functions apply to channels that have been opened to files.

When supported by the underlying operating system, the fopen() function performs security checks to determine whether the user name of the calling process has permission for the request.

If the fopen() function is successful, it returns the SMSL channel number to filename. A failure to open filename, such as an operating system problem or invalid mode, sets the SMSL errno value and causes the fopen() function returns the NULL string without attempting to open the file.

Support for binary file access

The fopen() function permits binary modes with a b character though there is no way within a SMSL process to write any form of binary data other than character strings.

Flush a file after each SMSL file operation

The SMSL functions ensure that a file is flushed after every operation so that the well-known bug of doing a write-then-read or read-then-write without an intervening fseek rewind or fflush does not occur in SMSL file operations.

fseek()

Set the file position indicator.

Format

fseek(*channel*,*offset*,*whence*)

Parameters

Parameter	Definition
channel	The file I/O channel returned when the file was opened by the fopen() function
offset	Number of bytes to be added to whence to obtain the file position
whence	Standard point within a file to which offset is added to obtain the new file position Valid range: One of the following integer values: 0 SEEK_SET, the beginning of the file 1 SEEK_CURR, the current file position 2 SEEK_END, the end of the file

Description

The fseek() function sets the filename position indicator to the whence position plus offset bytes. If whence is invalid, the fseek() function defaults to whence = 0 and raises a run-time error but completes the file seek operation.

NOTE 2 – Issuing the fseek() function against binary files with whence = 2 (SEEK_END) is not meaningfully supported on all platforms.

The fseek() function returns 0 for success and –1 for failure. For an invalid channel, that is, for a pipe channel instead of a file channel, the fseek() function returns –1, raises a run-time error and sets the SMSL errno variable.

fseek and append file mode

Using the fseek() function to change the file position indicator in a file opened in append mode; that is, modes a, ab or a+ will not prevent writes to the end of the file using the write() function.

Example

SMSL contains no equivalent to the C-rewind() function, but the following fseek() function example is the equivalent of the C-function rewind(*channel*):

fseek(channel,0,0);

ftell()

Return the file position indicator.

Format

ftell(*channel*)

Parameter

Parameter	Definition
channel	The file I/O channel returned when the file was opened by the fopen() function

Description

The ftell() function returns the file position indicator as the integer number of bytes from the beginning of the file. For an invalid channel, that is, a pipe channel instead of a file channel, the ftell() function returns –1, raises a run-time error, and sets the SMSL errno variable.

The typical result of both the C and SMSL versions of the ftell() function is the number of characters written to or read from a file, except on those platforms that perform CR/LF Û new-line conversions on text files. However, the value of the ftell() function after executing an fseek() function to the end of file is usually the total number of characters in the file.

The following SMSL functions change the file position indicator:

- fopen();
- fseek();
- read();
- readln();
- write().

The get_chan_info() function does not change the file position indicator. The close() function makes *channel* invalid.

full_discovery()

Verify that the process is currently in a full discovery cycle.

Format

full_discovery()

Description

The full_discovery() function returns TRUE if the SMSL script containing it is an application discovery script and it is currently in a full discovery cycle. Otherwise, the full_discovery() function returns FALSE.

A full discovery cycle is done after the agent's process cache is refreshed. This flag therefore indicates whether the process cache has been refreshed since the last time the script was executed.

Example

The following example tests whether the SMSL script is in a full discovery cycle and exits the script if it is not.

```
# If we are not in a full discovery cycle
# we can exit immediately
if (!full_discovery())
{
    exit;
}
```

get()

Return the current value of a variable.

Format

get(*variable*)

Parameter

Parameter	Definition
variable	Name of the variable whose current value will be returned

Description

The get() function returns the current value of variable. If *variable* is a relative name and does not exist in the context of the SMSL script, the get() function successively searches each ancestor's context until *variable* is found or until the search fails in the context of the computer.

Example

The following example returns the current status of RDB database Dev.

```
get ("/RDB/Dev/status");
```

get_chan_info()

Return status information from a SMSL file or process channel.

Format

get_chan_info(*channel*,*flags*)

Parameters

Parameter	Definition
channel	Channel name (shared channels) or number (local channels) whose status should be reported or "" indicating all channels should be reported (subject to flags control)
flags	Integer value representing two binary flags that controls the output of global and local channel information as follows: 1 global channels only 2 local channels only 3 both global and local channels Default if not specified: 1

Description

The get_chan_info() function returns channel information a string with the format

name status details type scope read_pid read_name write_pid write_name

Specifying:

get_chan_info("");

causes the get_chan_info() function to return descriptions for all global shared channels. The descriptions are formatted as a new-line separated list, one line per channel.

Field definition

name One of the following:

- scope=SHARED – Channel name.
- scope=LOCAL – Local channel number.
- scope="" – All shared and local channels.

status OPEN or CLOSED

details One of the following:

- fopen() channel – File name that is opened or NONE if no file name is open;
- popen() channel – Process ID of the external operating system process to which the channel is attached; or
- –1 if the process has terminated.

type PIPE or FILE

scope SHARED or LOCAL

read_pid One of the following:

- Process ID of the SMSL process waiting to read from the channel.
- –1 if no process is waiting.

read_name One of the following:

- Name of the process waiting to read from the channel.
- NONE if no process is waiting.
- UNAVAILABLE if there is a process but the name is not available.

write_pid One of the following:

- Process ID of the SMSL process waiting to write to the channel.
- –1 if no process is waiting.

write_name One of the following:

- Name of the process waiting to write to the channel.
- NONE if no process is waiting.
- UNAVAILABLE if there is a process but the name is not available.

Specifying

get_chan_info("",flags);

causes the get_chan_info() function to return all the local or global channels for the current SMSL process as controlled by the value of flags. Note that the following get_chan_info() functions are equivalent, for both return the list of global channels:

get_chan_info("");

get_chan_info("",1);

The get_chan_info() function produces a run-time warning if flags is non-numeric or not greater than zero, but the SMSL variable errno is not set to any value. The SMSL interpreter ignores flags without error if channel is not the empty string.

The get_chan_info() function returns all the fields for each channel even if they do not apply to the particular channel.

The get_chan_info() function returns the NULL string if:

- there are no global shared and/or local channels for the given value of flags;
- it receives a bad channel number or name.

In this case, the get_chan_info() function also sets the SMSL errno variable.

getenv()

Return the string value of a SMSL environment variable.

Format

getenv(*variable*)

Parameter

Parameter	Definition
variable	Name of the object whose value is to be returned

Description

The getenv() function returns the string value of variable in the environment of the SMSL script. The variable value can be returned from any of the following places:

- the parameter's defined environment variables;
- the application's defined environment variables;
- the computer's defined environment;
- the environment of the Agent at the start of its execution.

The getenv() function searches the environment tables in stated order and returns the value of the first matching variable. The getenv() function returns the NULL string if variable is not defined and sets the SMSL errno variable to a non-zero value. If the getenv() function is successful, it returns the value of variable and sets the SMSL errno variable to zero. Hence, you can use errno to distinguish an undefined variable from one that is set to the NULL string.

Example

This SMSL example presents a function that tests whether an environment variable exists.

```
function is_environment_var_defined(name)
{
    getenv(name); # Throw away return value of getenv
    return (errno == 0); # errno is only zero if name is defined
}
```

get_vars()

Return the list of variables for a SMSL object.

Format

get_vars(*object*,*showchildren*)

Parameters

Parameter	Definition
object	Optional name of the object whose variables are to be listed Default if not specified: Current object
showchildren	Optional flag whose non-zero value indicates get_vars() should also list the subobjects of object. Default if not specified: 0

Description

The get_vars() function returns a list of the variables of object or for the current object if object is omitted. The get_vars() function returns the NULL string if object does not exist.

The list of object variables is sorted in ascending alphabetical order.

grep()

Return the lines from a text block that match a regular expression.

Format

grep(*regular_expression*,*text*,*v*)

Parameters

Parameter	Definition
regular_expression	Character sequence that defines the pattern that the grep function searches for in text. *regular_expression* conforms to the regular expressions defined in the UNIX ed(1) command and the UNIX regexp(5) description. Following is a brief summary of several regular expression characters: ^ beginning of line \< beginning of a word $ end of line \> end of a word . match any single character * match zero or more repetitions of the preceding [] match any of the characters contained within [^] match any characters except those contained within.
text	Text to be searched for matches to regular_expression. text can be a text string enclosed in double quotation marks, or one or more SMSL commands that produce text as output.
v	The character v reverses the output of the grep() function, causing the grep() function to output all lines in text that do not contain a match for regular_expression. This flag is similar to the UNIX grep -v flag.

Description

The grep() function returns a list of the lines in text that match *regular_expression*.

If a character other than v is submitted as the grep() function reverse matching flag, the SMSL interpreter returns a run-time error message.

Example

```
# search for "martin" substring in /etc/passwd
all_lines=cat("/etc/passwd");
# fill a buffer with passwd
matching_lines=grep("martin",all_lines);
```

history()

Return history information from the history database.

Format

history(*parameter*,*format*,*number*)

Parameters

Parameter	Definition
parameter	Name of the object whose history should be returned. The expression “” indicates the current parameter. parameter can be: the absolute path, such as "/APP/INST/PARAM" a relative path, such as "." or "../DIFFERENTINST/PARAM". parameter can be "" or "." for the current parameter’s history. Default if not specified: Current parameter
format	Optional character string inside double quotation marks that specifies the format of each history() function entry. Valid Values: n return the number of available data points as the first value in the return list t include the time stamp of each entry in the return list v include the value of each history entry in the return list Default if not specified: ntv
number	Optional numeric value that limits the number of entries the history function will return. Default if not specified: 50

Description

The history() function accesses the parameter history database and returns a list containing the number of data points available followed by a number of entries.

The history() function returns the empty string, produces a run-time error, and sets the SMSL errno variable if a bad format character is provided.

Because of the defaults provided in the history() function, the following function specifications are equivalent:

```
history(parameter)
history(parameter,"ntv",50)
```

History output format

The history() function will return any of the following formats, depending on which format flags are set:

- *number_entries*\n if the n flag is set;
- value\n, time\n, or value time\n if the v, t, and vt flags are set.

The history() function separates the values of an entry with spaces and successive entries with new-line characters.

You can use the nthline(list1) function to get the number of points from the head of the list and also to extract the entries. Entries can be split if necessary into time and data values using the ntharg() function. The entries will be single values if either the t or v flag is absent.

index()

Return the starting position of one string within another.

Format

index(*text*,*string*)

Parameters

Parameter	Definition
text	Text to be searched for the occurrence of string. *text* can be a text string enclosed in double quotation marks, or one or more SMSL commands that produce text as output.
string	One or more characters enclosed in double quotation marks that are to be located within text

Description

The index() function returns the position in text at which string begins, or 0 if string does not occur in text. The first position in text is position 1.

int()

Return the largest integer that is not greater than the argument.

Format

int(*number*)

Parameter

Parameter	Definition
number	Numeric value or numeric variable

Description

The int() function returns the largest integer that is not greater than *number*.

internal()

Process a command internal to the Agent.

Format

internal(*command*)

Parameter

Parameter	Definition
command	Text string that is the command the Agent should process.

Description

The internal() function causes the Agent to process the string command internally and in a platform-specific manner. The internal() function returns the command output if successful. If unsuccessful or if the command is not supported on the specific platform, the internal() function returns the NULL string and sets the SMSL variable errno to E_SMSL_NOT_SUPPORTED.

The internal() function is designed to be used for user and process monitoring and resource inquires that can be handled inside the Agent. In these cases, the internal() function is much more efficient than invoking a separate command that requires a call to the SMSL interpreter or some other command processor.

intersection()

Return a list containing elements that are common to all specified lists.

Format

intersection(*list1,list2,list3list4 . . . listn*)

Parameters

Parameter	Definition
list1 . . . listn	Two or more SMSL lists that are being evaluated for common elements

Description

The intersection() function returns a SMSL list containing the elements that appear in all the lists *list1 . . . listn.*

The returned list is not well-defined and will contain duplicates if duplicates were present in all lists in the same number and order. The elements in the list returned by the intersection function appear in the same order as they were in list1.

If any lists are the NULL list, the return value is the NULL list; otherwise all entries in the returned list are terminated by a new-line character.

isnumber()

Verify that a string is a valid numeric representation.

Format

isnumber(*string*)

Parameter

Parameter	Definition
string	String that is to be evaluated as meeting the criteria for a numeric expression

Description

The isnumber() function returns a Boolean value of 1 if variable is a string that is considered valid as a number or "0" if it is not.

A valid number has only digits, periods, or minus signs for every character in variable. White space or any other invalid character anywhere in the string causes the isnumber() function to return 0. The isnumber() function returns 0 for the NULL string.

is_var()

Verify that a SMSL object variable exists.

Format

is_var(*object*)

Parameter

Parameter	Definition
object	Name of the object that is to be verified as a variable

Description

The is_var() function returns TRUE if object exists and is a variable. The is_var() function returns FALSE if:

- object does not exist;
- object exists but is not a variable (that is, it is an application instance or a parameter).

length()

Return the number of characters in a string.

Format

length(*text*)

Parameter

Parameter	Definition
text	Text to be counted for character length. *text* can be a text string enclosed in double quotation marks, or one or more SMSL commands that produce text as output.

Description

The length() function returns the length in characters of text, including new lines.

lines()

Return the number of lines in a string.

Format

lines(*text*)

Parameter

Parameter	Definition
text	Text to be counted for number of lines (that is, new-line characters). text can be the name of a text file, a text string enclosed in double quotation marks, or one or more SMSL commands that produce text as output.

Description

The lines() function returns the number of new-line characters in text. The lines() function is useful for returning the length of a list because the items in a list are delimited by new lines.

lock()

Acquire a SMSL process lock.

Format

lock(*lockname*,*mode*,*timeout*)

Parameters

Parameter	Definition
lockname	Name of the lock that should be acquired
mode	Optional control permitted under the lock. Valid range: s shared r reader w writer x exclusive Default if not specified: x If the first letter of mode is not s, r, w, or x a run-time error occurs and mode defaults to x (exclusive).
timeout	Optional integer value that specifies the number of seconds before the lock request expires. Valid range: timeout > 0 is the integer number of seconds the lock request is valid. timeout = 0 means non-blocking lock request timeout < 0 means infinite timeout period (that is, wait until the lock is released) Default if not specified: Infinite timeout

Description

The lock() function requests a lock with name *lockname*. The mode of the request specifies either shared (*reader*) or exclusive (*writer*) access under the lock. The optional timeout specifies the number of seconds the request is valid.

The default behaviour of lock() function is to request an exclusive lock with an infinite timeout period. The lock() function returns 1 for success and 0 for failure.

Locks and SMSL

Lock names are global to the Agent; thus:

- all SMSL processes share the same table of locks;
- different SMSL processes can share parameter lock names to perform concurrent actions.

There is no way to enforce lock naming scope. It is recommended that lock names in SMSL programs be uniquely encoded using the name of the application. This practice will avoid potential clashes with other SMSL programs.

Shared lock requests

Shared lock requests for a lock that is currently in share mode are granted – unless there is a waiting write request. Giving priority to a waiting write request prevents the lock mechanism from starving write processes.

Requests for locks already held

It is possible to request a lock that you already hold although it is not good style:

- requesting a lock that you already hold is ignored;
- requesting a shared lock on a lock you already hold with exclusive access is also ignored.

Requesting an upgrade to exclusive access of a lock currently held as shared succeeds and upgrades the lock provided you are the only process that is using the shared lock. If you are not the only process using the lock, the lock() function immediately returns 0 in non-blocking mode (regardless of the value of timeout because blocking would cause immediate deadlock by waiting for yourself!).

Rather than using this upgrade feature, it is recommended that you call the unlock() function to release the shared lock before attempting to acquire the new exclusive lock. Lock tracing is possible using the SMSLDebug variable. SMSLDebug can be useful in debugging multiprocess lock interactions.

Failure of the lock function

The lock() function can fail if:

- a non-blocking request fails;
- timeout is exceeded before the lock is granted.

The lock() function can fail for an infinite timeout if:

- a special-case upgrade request is granted;
- the system has performed some external deadlock correction.

loge()

Return the natural logarithm of the argument.

Format

loge(*argument*)

Parameter

Parameter	Definition
argument	Numeric value whose natural logarithm is to be determined. Valid range: argument > 0

Description

The loge() function returns the logarithm of argument with respect to the natural logarithm base e = 2.71828 . . . The output range for the loge() function is $-\infty < \text{loge}() < \infty$.

log10()

Return the logarithm to base 10 of the argument.

Format

log10(*argument*)

Parameter

Parameter	Definition
argument	Numeric value whose base 10 logarithm is to be determined. Valid range: argument > 0

Description

The log10() function returns the logarithm of argument with respect to base 10.

The output range for the log10() function is $-\infty < \log10() < \infty$.

ntharg()

Return a formatted list containing fields from a text string.

Format

ntharg(*text,arguments,delimiters,separator*)

Parameters

Parameter	Definition
text	Text to be separated into fields by the ntharg() function. *text* can be a text string enclosed in double quotation marks, or one or more SMSL commands that produce text as output.
arguments	Integer list specifying the field numbers ntharg() should look for in each line of text. Fields are specified as follows: *x,y* field x and field y x-y all fields from x to y inclusive -x all fields from 1 to x inclusive x- all fields from x to the new-line character inclusive
delimiters	One or more characters that ntharg() should treat as field separators when examining text. Default if not specified: space and \t (*tab*)
separator	Optional character that should be placed between each field of ntharg() output Default if not specified: new-line character ()

Description

The ntharg() function returns the arguments in text.

The ntharg() function normally interprets each line in text as a white space-separated (space or tab) list of fields. If delimiters is given, it specifies the list of characters that ntharg() should treat as field separators. The ntharg() function normally returns selected fields as a new-line delimited list. If separator is given, it specifies the delimiter to be placed between items in the returned list.

NOTE 3 – The difference between the ntharg() function and the nthargf() function is as follows:

- The ntharg() function treats each delimiter that follows a non-delimiter character as the end of a field. The ntharg() function interprets two or more adjacent delimiters as a single delimiter.
- The nthargf() function treats each delimiter as the end of a field. The nthargf() function interprets two or more adjacent delimiters as delimiting one or more NULL strings whose content can be requested and returned.

Example

The following example prints the login name and home directory of each user listed in the UNIX system password file.

```
 foreach user (cat("/etc/passwd"))
{
    printf(ntharg(user,"1,6",":","\t"),"\n");
}
```

nthargf()

Return a formatted string containing fields from a text string.

Format

nthargf(*text,arguments,delimiters,separator*)

Parameters

Parameter	Definition
text	Text to be separated into fields by the nthargf() function. *text* can be a text string enclosed in double quotation marks, or one or more SMSL commands that produce text as output.
arguments	Integer list specifying the field numbers nthargf() should look for in each line of text. Fields are specified as follows: *x,y* field x and field y x-y all fields from x to y inclusive -x all fields from 1 to x inclusive x- all fields from x to the new-line character inclusive
delimiters	One or more characters that nthargf() should treat as field separators when examining text. The nthargf() function treats each occurrence of delimiters as delimiting a field. The nthargf() function interprets two or more adjacent delimiters as delimiting one or more NULL fields. Default if not specified: space and \t (*tab*)
separator	Optional character that should be placed between each field of nthargf() output Default if not specified: new-line character ()

Description

The nthargf() function returns the arguments in text.

The nthargf() function normally interprets each line in text as a white space-separated (space or tab) list of fields. If delimiters is given, it specifies the list of characters that nthargf() should treat as field separators. The nthargf() function normally returns selected fields as a new-line delimited list. If separator is given, it specifies the delimiter to be placed between items in the returned list.

NOTE 4 – The difference between the nthargf() function and the ntharg() function is as follows:

- The nthargf() function treats each delimiter as the end of a field. The nthargf() function interprets two or more adjacent delimiters as delimiting one or more NULL strings whose content can be requested and returned.
- The ntharg() function treats each delimiter that follows a non-delimiter character as the end of a field. The ntharg() function interprets two or more adjacent delimiters as a single delimiter.

nthline()

Return specified lines from a text string.

Format

nthline(*text,lines,separator*)

Parameters

Parameter	Definition
text	Text to be separated into lines by the nthline() function. *text* can be a text string enclosed in double quotation marks, or one or more SMSL commands that produce text as output.
lines	Integer list specifying the line numbers nthline() should look for in text. Lines are specified as follows: *x,y* line x and line y x-y all lines from x to y inclusive -x all lines from 1 to x inclusive x- all lines from x to EOF character inclusive
separator	Optional character that should be placed between each field of nthline() output Default if not specified: new-line character ()

Description

The nthline() function returns the lines of text separated by new-line characters. If you specify a separator, the nthline() function will use separator to separate lines.

NOTE 5 – The difference between the nthlinef() and nthline() functions is as follows:

- The nthlinef() function treats each new-line character as a line.
- The nthline() function treats only a non-empty line (that is, a line with a non-new-line character preceding a new-line character) as a line.

Example

The following SMSL script prints the top five processes executing on a UNIX system.

```
# print the top five processes
printf(“%s”, nthline(system("ps -eaf"),"2-6"));
```

nthlinef()

Return specified lines from a text string.

Format

nthlinef(text,lines,separator)

Parameters

Parameter	Definition
text	Text to be separated into lines by the nthlinef() function. *text* can be a text string enclosed in double quotes, or one or more SMSL commands that produce text as output.
lines	Integer list specifying the line numbers nthlinef() should look for in text. Lines are specified as follows: *x,y* line x and line y, *x-y* all lines from x to y inclusive, *-x* all lines from 1 to x inclusive, *x-* all lines from x to EOF character inclusive
separator	Optional character that should be placed between each field of nthlinef() output Default if not specified: new-line character ()

nthlinef()

Return specified lines from a text string.

Format

nthlinef(text,lines,separator)

Parameters

Parameter	Definition
text	Text to be separated into lines by the nthlinef() function. *text* can be a text string enclosed in double quotes, or one or more SMSL commands that produce text as output.
lines	List specifying the line numbers nthlinef() should look for in text. Lines are specified as follows: *x,y* line x and line y, x-y all lines from x to y inclusive, -x all lines from 1 to x inclusive, x- all lines from x to EOF character inclusive Integer
separator	Optional character that should be placed between each field of nthlinef() output Default if not specified: new-line character ()

Description

The nthlinef() function returns the lines of text separated by new-line characters. If you specify a separator, the nthlinef() function will use separator to separate lines.

NOTE 6 – The difference between the nthlinef() and nthline() functions is as follows:

- The nthlinef() function treats each new-line character as a line.
- The nthline() function treats only a non-empty line (that is, a line with a non-new-line character preceding a new-line character) as a line.

It is recommended that you use nthlinef() function to be consistent with other SMSL functions.

Example

The following SMSL script prints the top five processes executing on a UNIX system.

```
# print the top five processes
printf(“%s”,nthlinef(system("ps -eaf"),"2-6"));
```

popen()

Open a SMSL channel to a process.

Format

popen(*type,command,instance*)

Parameters

Parameter	Definition
type	Command processor that should interpret and execute command Valid range: The built-in command types OS or SMSL, or a valid user-defined command type
command	Syntax of the submitted command
instance	The application instance against which command should execute Default if not specified: The application instance that is the nearest ancestor of command.

Description

The popen() function spawns a process to execute a command of a defined type and returns a channel number which can then be used to read the command’s output or write messages to the command.

The popen() function returns –1 on error.

pow()

Raise a number to a power.

Format

pow(*base,exponent*)

Parameters

Parameter	Definition
base	Numeric value that is to multiplied by itself *exponent* number of times
exponent	Numeric value that indicates the number of times *base* should be multiplied by itself. *exponent* must be positive if *base* ≥0, and *exponent* must be an integer if base < 0.

Description

The pow() function returns the value of base raised to the power exponent, or base exponent.

The output range for the pow() function is $-\infty < pow() < \infty$.

printf()

Print text formatted to the C-library printf() routine specification.

Format

printf(*format*)

Parameter

Parameter	Definition
format	Text, variable names, and control characters that specify the content and format of output to the computer or task output window. format permits the following conversion characters: %d signed decimal (identical to %i) %i signed decimal (identical to %d) %u unsigned decimal %o unsigned octal %x unsigned hexadecimal using abcedf %X unsigned decimal using ABCDEF %c unsigned character %s character string %e double precision form drddde±ddd where each d is a digit and r is the radix character %E double precision form drdddE±ddd where each d is a digit and r is the radix character %f decimal notation form dddrddd where each d is a digit and r is the radix character %g print in the style of %e if the exponent after conversion is less than –4, else print in style %f %G print in the style of %E with the precision specifying the number of significant digits %N Group digits into threes and separate with commas beginning at the right of the string %% print a % character format does not support the standard C-pointer conversion characters %p and %n. format permits the following flags: left-justify and pad on the right with spaces + display plus sign when value is grater than zero 0 pad with zeros if no other padding is specified # alters the meaning of a conversion: appends 0x or 0X to the %x and %X conversions always appends the radix character to the %e, %E, %f, %g, and %G conversions retains trailing zeros in the %g and %G conversions The # flag does not affect the %c, %d, %s, or %i conversions

Description

The printf() function displays output to the computer or task output window using formatting similar to the standard C-printf() function.

A bad format or one that is valid for the C language but not for the printf() function results in a SMSL run-time error that sets the SMSL errno variable.

The printf() function return value is always the null string.

C conventions not supported by the SMSL printf function

The printf() function does not support the C convention of using the asterisk (*) as a field width or precision indicator. The printf() function does not support the %p and %n conversion characters.

The length modifiers h, l (*ell*), and L are not valid and are ignored by the printf function.

The printf() function format conversions are passed directly to the C-library printf() routine on each platform. The output for obscure formatting features may differ across platforms.

Conversion differences between the C printf routine and SMSL printf function

The format conversions have the same meaning between standard C and SMSL, but the concept of variable types differs between the two.

SMSL supports only string types for its variables, and thus string arguments to the printf() function are converted in a manner appropriate for the format conversion:

- Integral formats such as %d convert the string to signed integers.
- Non-integer numeric formats such as %f convert to floating point values.

- %c prints the ASCII equivalent of its integer argument or for non-numeric arguments the first character of its argument. (Applying %c to "65" will print 'A' and to "AB" will print 'A'.)
- %s causes no conversion.
- %% requires no argument.

The %N Format Conversion

The printf() function provides one non-standard C extension – the %N conversion. The %N conversion preprocesses a numeric string so that commas separate each group of three digits beginning at the right side of the string.

For example, the %N conversion causes the following conversions:

$$1234 \Rightarrow 1{,}234 \quad 12345 \Rightarrow 12{,}345 \quad 123456 \Rightarrow 123{,}456$$

The %N conversions ignores initial minus signs and blanks while searching for the first sequence of digits so that %N can be applied to negative values. If no digits are found after the skipped characters, the printed argument is unchanged.

The %N conversion only modifies the first sequence of digits. For example, the %N conversion changes floating point numbers like 1234.1234 to become 1,234.1234 without changing to the digit sequence to the right of the decimal point.

As part of the %N conversion, the printf() function performs a %s conversion using the field width and precision specifiers supplied in format. The printf() function prints the string resulting from the combined %N and %s conversions. Because of the embedded %s conversion, field width and precision under %N conversion have the same effect as with %s.

NOTE 7 – Currently, no localization is supported by %N, and so the formatting achieved by %N does not change in different locales.

proc_exists()

Verify that a process exists.

Format

proc_exists(*pid*)

Parameter

Parameter	Definition
pid	Process identifier number of the process whose existence is being verified

Description

The proc_exists() function returns TRUE if the process with process identifier pid exists; FALSE if it does not.

process()

Return a list of processes from the Agent process cache.

Format

process(*regular_expression*)

Parameter

Parameter	Definition
regular_expression	Character sequence that defines the pattern the process() function searches for in the Agent process cache. *regular_expression* conforms to the regular expressions defined in the UNIX ed(1) command description and the UNIX regexp(5) description. Following is a brief summary of several regular expression characters: ^ beginning of line \< beginning of a word $ end of line \> end of a word . match any single character * match zero or more repetitions of the preceding [] match any of the characters contained within [^] match any characters except those contained within

Description

The process() function returns the list of processes in the Agent's process cache that match the regular expression regular_expression. Each entry in the list is a string formatted as follows:

pid ppid user status size cputime *command_name command_line*

NOTE 8 – Some platforms do not support all the return values. For a specific platform, the process() function generally returns the same information as the ps command. The process() function returns the NULL string if no processes match regular_expression.

Example

The following SMSL commands list all ORACLE database process daemons.

```
# find ORACLE database daemons
ora_procs = process("ora_");
printf ("%d", ora_procs);
```

Parameter

Parameter	Definition
pid	Process identifier number
ppid	Parent process identifier number
user	User name to which the process belongs
status	Process status within the system. Valid range: 0 Non-existent S Sleeping W Waiting R Running I Intermediate Z Terminated T Stopped X Growing
size	Process core image size (in blocks)
cputime	Integer number of CPU seconds consumed by the process
command_name	First word of the command line that started the process
command_line	Complete command line that started the process. Note that the command line may have been modified during process execution.

random()

Return a random number.

Format

random(*maximum*)

Parameter

Parameter	Definition
maximum	Valid range: maximum > 0 Default if not specified: maximum = $2^{32} - 1$ (from the underlying C function)

Description

The random() function is equivalent to the standard C-library random() function.

If maximum is zero or negative, the random() function will return a run-time error message. Optional upper bound for the values returned by random:

$$0 \leq \text{random}() \leq \text{maximum} - 1$$

read()

Read from a SMSL file or process channel.

Format

read(*channel*,*size*)

Parameters

Parameter	Definition
channel	The process I/O channel number from which the read() function is to read data
size	Integer value controlling the amount of data that the read() function will read from channel. Valid Range: size > 0 instructs the read() function to read at least size bytes and return size = 0 instructs the read() function to return as soon as it has read something from the channel size = –1 instructs the read() function to read all data available from the channel and return Default if not specified: size = 0

Description

The read() function returns the data it reads from channel. The read() function returns the value EOF (that is, the NULL string) on an end-of-file or error condition.

Channels are created by calling the fopen() or popen() function.

NOTE 9 – The read function can block for a process channel created using the popen() function but not for a file channel created using the fopen() function.

To enforce serialization for shared channels, no two reader processes (that is, read() or readln() functions) can be blocked on the same channel. The second reader process that attempts to block on the shared channel will fail, returning the NULL string and setting the SMSL variable errno to E_SMSL_BUSY_CHANNEL.

Another possible shared channel failure can be caused by a close() function being executed against a channel that also has a blocked reader process. The close() function will cause the reader process to return the NULL string and set errno to E_SMSL_UNBLOCKED_BY_CLOSE.

Example

The following SMSL example opens a channel to the UNIX operating system, executes a UNIX ls command, then reads and prints the directory entries returned by the ls command.

```
chan = popen ("OS", "ls");
while ((data = read (chan)) ! = EOF)
{
    printf("%s", data);
}
close (chan);
```

readln()

Read a line of data from a SMSL file or process channel.

Format

readln(*channel*)

Parameter

Parameter	Definition
channel	The process I/O channel number from which the readln function is to read data

Description

The readln() function reads the next line of data from channel and returns it. The readln() function returns the value EOF (NULL) on end-of-file or error.

Channels are created by calling the fopen() or popen() function. Note The readln() function can block for a pipe channel created using the popen() function but not for a file channel created using the fopen() function.

To enforce serialization for shared channels, no two reader processes (that is, read() or readln() functions) can be blocked on the same channel. The second reader process that attempts to block on the shared channel will fail, returning the NULL string and setting the SMSL variable errno to E_SMSL_BUSY_CHANNEL.

Another possible shared channel failure can be caused by a close() function being executed against a channel that also has a blocked reader process. The close() function will cause the reader process to return the NULL string and set errno to E_SMSL_UNBLOCKED_BY_CLOSE.

Limitation

The readln() function has a line limitation of 4K when executed against files opened with the fopen() function. The readln() function may truncate lines longer than 4K. This limitation does not apply to channels opened using the popen() function.

rindex()

Return the last occurrence of one text string within another.

Format

rindex(*text,string*)

Parameter

Parameter	Definition
text	Text to be examined for occurrences of string. *text* can be a text string enclosed in double quotation marks, or one or more SMSL commands that produce text as output.
string	One or more characters whose last occurrence is being identified within text

Description

The rindex() function returns the position of the last occurrence of string in text or 0 if string does not occur in text. Positions in string are numbered starting from one.

set()

Assign a value to a variable.

Format

set(*variable,value*)

Parameters

Parameter	Definition
variable	The name of a variable in the Agent object hierarchy to which value is assigned
value	The numeric or string value that is assigned to variable

Description

The set() function sets the value of variable to be value. If variable is a relative name and does not exist in the context of the SMSL script, the set() function successively searches each ancestor's context until variable is found or until the search fails in the context of the computer.

The set() function returns value if the assignment is successful, the NULL string if it is not.

Example

The following SMSL statement sets the value of RDB database Dev parameter MyParam to 10.

```
set("/RDB/Dev/MyParam/value",10);
```

share()

Convert a local channel into a shared global channel.

Format

share(*channel,name*)

Parameters

Parameter	Definition
channel	Process I/O channel number that was returned when the channel was opened using the fopen() or popen() function
name	Character string used to identify the shared channel in the table of global channels It is recommended that you specify a non-numeric name to avoid conflicts with numbers used internally for local channels. Using a number for name does not actually cause the share() function to fail but will raise a SMSL run-time warning. The share() function will dutifully place the specified numeric name in the global table, leading to potential conflicts with local channels in close(), read(), write(), and readln() functions.

Description

The share() function is the main function for using shared channels. The share() function propagates an existing local channel into the table of global channels as name. Channels opened by either the popen() or fopen() functions can be shared.

If the share() function is successful, it returns 0. The local channel is no longer available in the process that opened it and does not require a close() function. In fact, the close() function will return an error since it will not find the local channel.

The share() function will fail, returning –1 and setting the SMSL errno variable if:

- the local channel does not exist;
- the global channel name already exists in the global channel table.

Upon failure, the local channel is unchanged and still available. No global channel is added.

A global channel is referred to by name when passed to the read(), readln(), write(), and close() functions. These functions will first search the local channel table containing only channel numbers and then the global channel table.

sin()

Return the sine of the argument.

Format

sin(*radians*)

Parameter

Parameter	Definition
radians	Arc length in radians whose sine is to be determined Valid range: $-\infty \leq$ radians $\leq \infty$

Description

The sin() function returns the sine of radians. The output range for the sin() function is $-1 \leq \sin() \leq 1$.

sinh()

Return the hyperbolic sine of the argument.

Format

sinh(*argument*)

Parameter

Parameter	Definition
argument	Numeric value whose hyperbolic sine is to be determined Valid range: $-\infty \leq$ argument $\leq \infty$

Description

The sinh() function returns the hyperbolic sine of argument. The hyperbolic sine is defined by the expression:

$$\sinh(x) = (e^{x} - e^{-x})/2$$

where *e* is the base for the natural logarithms (e = 2.71828 . . .). The output range for the sinh() function is $-\infty \leq \sinh() \leq \infty$.

sleep()

Suspend process execution for a number of seconds.

Format

sleep(*seconds*)

Parameter

Parameter	Definition
seconds	Integer specifying the number of seconds the process should be suspended. Valid range: seconds > 0 is the number of seconds the process will sleep seconds ≤ 0 the timer expires immediately

Description

The sleep() function suspends a SMSL process for the specified number of seconds. While suspended, the SMSL process consumes no CPU resources and is not interpreted until awakened by the expiration of the seconds timer.

NOTE 10 – The sleep() function only suspends the process that calls it. All other SMSL processes continue normal execution.

The sleep() function returns a run-time warning if seconds is non-numeric, in which case the timer defaults to zero.

sort()

Sort a list of numeric or alphabetic values.

Format

sort(*list,mode,position*)

Parameters

Parameter	Definition
list	SMSL list whose elements are to be sorted
mode	Optional character string specifying the sort order. Character string must be enclosed in double quotation marks. Valid Range: "n" ascending numeric order; "nr" descending numeric order; "" ascending alphabetic order; "r" descending alphabetic order Default if not specified: “” (ascending alphabetic)
position	Optional integer that specifies the character position within each element of list where sorting is to begin The first character of each list element is character 1. If the length of every element within list is less than position, the effect is the same as if all the elements of list were NULL elements. position does not truncate elements; it only ignores the first (position –1) characters for purposes of comparison. Default if not specified: 1

Description

The sort() function returns a sorted version of list that is ordered according to mode.

The sort() function does not merge duplicate entries in list: the returned list has the same number of members as list. The order in which duplicates are returned is not defined because it is not defined for the C-library qsort() function. This fact is relevant for the following cases:

- numeric sorting of strings with identical numeric prefix values but different non-numeric suffixes;
- any sorting mode in which position is larger than more than one element within list (the sort() function regards all such elements as duplicate NULL elements).

If list is the NULL list, the sort function returns the NULL list. For a non-empty list, the sort() function always returns a well-defined list with the last line properly terminated by a new-line character.

NOTE 11 – List need not be terminated by a new-line character. Numeric sorting is based on floating point values; non-numeric list entries are converted according to the system's standard C-library function atof().

sprintf()

Return the specified format as a character string to a destination.

Format

sprintf(*format*)

Parameter

Parameter	Definition
format	Text, variable names, and control characters that specify the content and format of the character string output to the computer or task output window Format permits the following conversion characters: %d signed decimal (identical to %i) %i signed decimal (identical to %d) %u unsigned decimal %o unsigned octal %x unsigned hexadecimal using abcedf %X unsigned decimal using ABCDEF %c unsigned character %s character string %e double-precision form drddde±ddd where each d is a digit and r is the radix character %E double-precision form drdddE±ddd where each d is a digit and r is the radix character %f decimal notation form dddrddd where each d is a digit and r is the radix character %g print in the style of %e if the exponent after conversion is less than –4, else print in style %f %G print in the style of %E with the precision specifying the number of significant digits %N group digits into threes and separate with commas beginning at the right of the string %% print a % character format does not support the standard C-pointer conversion characters %p and %n. format permits the following flags: – left-justify and pad on the right with spaces + display plus sign when value is greater than zero 0 pad with zeros if no other padding is specified # alters the meaning of a conversion: appends 0x or 0X to the %x and %X conversions always appends the radix character to the %e, %E, %f, %g, and %G conversions retains trailing zeros in the %g and %G conversions The # flag does not affect the %c, %d, %s, or %i conversions.

Description

The sprintf() function is identical to the printf() function except that it returns the created string rather than outputting it. If there is an error in format, sprintf sets the SMSL errno variable and returns the NULL string.

The formats, conversions, and values of errno for the various errors are identical to those described for the printf() function.

C programmers should be careful to use the SMSL style:

destination=sprintf(*format*)

rather than the C style:

sprintf(*destination,format*)

The latter style will often cause a compilation warning about a null-effect statement.

C conventions not supported by the SMSL sprintf function

The sprintf() function does not support the C convention of using the asterisk (*) as a field width or precision indicator. The sprintf() function does not support the %p and %n conversion characters.

The length modifiers h, l (*ell*), and L are not valid and are ignored by the sprintf() function.

The sprintf() function format conversions are passed directly to the C-library sprintf() routine on each platform. The output for obscure formatting features may differ across platforms.

Conversion differences between the C-sprintf routine and SMSL sprintf function

The format conversions have the same meaning between standard C and SMSL, but the concept of variable types differs between the two.

SMSL supports only string types for its variables, and thus string arguments to the sprintf() function are converted in a manner appropriate for the format conversion:

- Integral formats such as %d convert the string to signed integers.
- Non-integer numeric formats such as %f convert to floating point values.
- %c prints the ASCII equivalent of its integer argument, or for non-numeric arguments, the first character of its argument. (Applying %c to “65” will print ‘A’ and to “AB” will print ‘A’.)
- %s causes no conversion.
- %% requires no argument.

sqrt()

Return the square root of the argument.

Format

sqrt(*argument*)

Parameter

Parameter	Definition
argument	Numeric value whose mathematical square root is returned Valid range: $-\infty \leq$ argument $\leq \infty$

Description

The sqrt() function returns the square root of the positive integer or real value argument.

srandom()

Initialize the random number generator with a seed value.

Format

srandom(*seed*)

Parameter

Parameter	Definition
seed	Numeric value used as a starting point for pseudorandom number generation by the random() function

Description

The srandom() function sets the random number seed for the random() function. *seed* is passed directly to the UNIX C srandom() function.

The SMSL srandom() function always returns the NULL string.

subset()

Verify that one SMSL list is a subset of another.

Format

subset(*set*,*subset*)

Parameters

Parameter	Definition
set	SMSL list, that is the set in the set-subset verification
subset	SMSL list, that is the subset in the set-subset verification

Description

The subset() function returns a Boolean value of 0 or 1 indicating whether subset is a proper or improper subset of set. If subset is the NULL set, the subset() function returns 1 (TRUE). If set is the NULL set and subset is not, the subset() function returns 0 (FALSE).

The subset() function ignores duplicates and returns 1 only if all elements of subset are also present in set.

Example

The subset() function can be used to determine whether a particular element is present in a set and thus provides “is_member” functionality such as the following:

```
if (subset(my_set,"blue"))
{
    # SMSL set "my_set" contains element "blue"
}
```

It is not necessary to place a new line at the end of the “blue” string because it is inserted by the subset() function. The example statements are treated as a subset() function acting on a set with one element.

substr()

Return a specified portion of a string of characters.

Format

substr(*text*,*start*,*length*)

Parameters

Parameter	Definition
text	Text from which a substring of characters is to be returned. *text* can be a text string enclosed in double quotation marks, or one or more SMSL commands that produce text as output.
start	The character position within text that is to be the first character of the substring. The first character in text is character position 1.
length	The total number of characters from text to be returned in the substring

Description

The substr() function returns the substring of text of length characters that starts at position *start*.

system()

Submit a command to the computer operating system.

Format

system(*command*,*instance*)

Parameters

Parameter	Definition
command	Syntax of the submitted operating system command. *command* can contain output redirection, pipes, wild cards, and so on.
instance	Optional application instance against which command should execute Default if not specified: The application instance that is the nearest ancestor of command

Description

The system() function returns any output produced by submitting command to the system-dependent command execution subsystem.

tail()

Return the last lines from a text block.

Format

tail(*text*,*lines*)

Parameters

Parameter	Definition
text	Text whose last lines are to be returned by tail(). *text* can be a text string enclosed in double quotation marks, or one or more SMSL commands that produce text as output.
lines	Number of lines of text to be returned, starting from the last line of text

Description

The tail() function returns the last lines number of lines of text.

tan()

Return the tangent of the argument.

Format

tan(*radians*)

Parameter

Parameter	Definition
radians	Arc length in radians whose tangent is to be determined Valid range: $-\infty \leq$ radians $\leq \infty$

Description

The tan() function returns the tangent of radians. The output range for the tan() function is $-\infty <$ tan() $< \infty$. The tan() function is undefined when radians = p(2n+1)/2 where n is an integer.

tanh()

Return the hyperbolic tangent of the argument.

Format

tanh(*argument*)

Parameter

Parameter	Definition
argument	Numeric value whose hyperbolic tangent is to be determined Valid range: $-\infty \leq$ argument $\leq \infty$

Description

The tanh() function returns the hyperbolic tangent of argument. The hyperbolic tangent is defined by the expression:

$$\tanh(x) = (e\ x - e\ -x\ )/(e\ x + e\ -x\ )$$

where e is the base for the natural logarithms (e = 2.71828 . . .). The output range for the tanh() function is $-1 \leq \tanh() \leq 1$.

time()

Return the number of seconds since 00:00:00 GMT January 1, 1970.

Format

time()

Description

The time() function returns the current time as the number of seconds that have elapsed since 00:00:00 GMT, Jan 01, 1970.

tmpnam()

Return a unique name for temporary file creation.

Format

tmpnam()

Description

The tmpnam() function returns a name that is guaranteed to be unique and can be used to pass to the fopen function for creating temporary files.

The semantics of the tmpnam() function are similar to that of the C tmpnam() routine – notably, a restricted number of unique names are returned by the tmpnam() routine as defined by the C-constant TMP_MAX. All SMSL processes on a given Agent share the same set of names, and there can be a danger of mixing names. If the size of TMP_MAX is a concern, add a suffix to the returned file name.

Example

The following example shows how to use the tmpnam() function to generate a temporary file name. The SMSL function adds a suffix to the returned name to further guarantee its uniqueness.

```
name = tmpnam() . ".dave"; fp = fopen(name, "w");
```

tolower()

Convert text to all lowercase characters.

Format

tolower(*text*)

Parameter

Parameter	Definition
text	Text that is to be returned as lowercase letters. *text* can be a text string enclosed in double quotation marks, or one or more SMSL commands that produce text as output.

Description

The tolower() function returns a copy of text with all uppercase letters converted to lowercase letters.

toupper()

Convert text to all uppercase characters.

Format

toupper(*text*)

Parameter

Parameter	Definition
text	Text that is to be returned as uppercase letters. *text* can be a text string enclosed in double quotation marks, or one or more SMSL commands that produce text as output.

Description

The toupper() function returns a copy of text with all lowercase letters converted to uppercase letters.

trim()

Remove unwanted characters from text.

Format

trim(*text,unwanted*)

Parameters

Parameter	Definition
text	Text to be returned without specified characters. *text* can be a text string enclosed in double quotation marks, or one or more SMSL commands that produce text as output.
unwanted	One or more characters that are to be removed from the copy of text output by the trim function.

Description

The trim() function returns a copy of text with all occurrences of the characters in unwanted removed.

union()

Return a list that is the union of individual lists.

Format

union(*list1,list2,list3,list4 . . . listn*)

Parameters

Parameter	Definition
listn	SMSL list containing elements that shall be united and returned in a single well-defined list. Only the first two input lists, list1 and list2, are required; all others are optional.

Description

The union() function returns a SMSL list that contains the elements from all listn merged together. Unlike the difference() and intersection() functions, the list returned by the union function is a well-defined set without any duplicates. The union() function adds a new line to the end of every non-empty list that is missing one. If the return value is not the NULL list, the returned set is always terminated by a new line so that all set elements end with a new-line character.

unique()

Remove the duplicate elements from a list.

Format

unique(*list*)

Parameter

Parameter	Definition
list	SMSL list containing elements that shall be returned in a single well-defined (*unique*) list

Description

The unique() function returns a well-defined SMSL list with all duplicates removed. All elements that remain in the return value appear in the same order as they did in list. If list is the NULL list, the unique() function returns the NULL list; otherwise the unique() function returns a list that is terminated by a new-line character so that all list elements in the list end with a new-line character.

unlock()

Release a SMSL process lock.

Format

unlock(*lockname*)

Parameter

Parameter	Definition
lockname	Name of the lock that should be released

Description

The unlock() function releases *lockname* that was granted to this process by a previous call to the lock() function. The unlock() function returns 1 for success and 0 for failure. If no lock is named *lockname* or if it is not currently owned by this process, then the unlock() function reports a run-time error, sets the SMSL errno variable, and returns 0.

NOTE 12 – All locks held by a process are automatically released when the process exits in a manner similar to executing the unlock function. It is recommended that you release locks explicitly using the unlock() function rather than implicitly using the process exit. If this process is the only one holding the lock and processes are queued for it, the first waiting process is awakened and granted use of the lock. If the first process is a shared request, then any other processes that are queued for a shared lock are also granted shared access to the lock (except for processes that are behind a writer request on the queue for this lock).

write()

Write to a SMSL process or file channel.

Format

write(*chan*,*text*)

Parameters

Parameter	Definition
chan	Process I/O channel number to which *text* is written
text	Text to be written to channel *chan*. *text* can be a text string enclosed in double quotation marks, or one or more SMSL commands that produce text as output.

Description

The write() function writes text to channel chan. The write() function returns the number of characters written or –1 on error.

If text cannot be written immediately, the write() function call blocks until it can either write the whole of text or the channel terminates.

NOTE 13 – The write() function can block for a process channel created using the popen() function but not for a file channel created using the fopen() function.

To enforce serialization for shared channels, no two reader processes (that is, the read() or readln() functions) can be blocked on the same channel. The second reader process that attempts to block on the shared channel will fail, returning the NULL string and setting the SMSL variable errno to E_SMSL_BUSY_CHANNEL.

Another possible shared channel failure can be caused by a close() function being executed against a channel that also has a blocked reader process. The close() function will cause the reader process to return the NULL string and set errno to E_SMSL_UNBLOCKED_BY_CLOSE.

Annex H

MOCS proforma

(This annex forms an integral part of this Recommendation | International Standard)

This annex contains MOCS proforma for the subset of object classes defined in [X721] that is used for SDH management.

The following common notations, defined in Recommendation X.724 are used for the status columns:

- m Mandatory
- o Optional
- c Conditional
- x Prohibited
- – Not applicable or out of scope

Note that “c”, “m”, “o” and “x” are prefixed by a “c:” when nested under a conditional or optional item of the same table.

Note that “o” may be suffixed by “n” (where “n” is a unique number) for mutually exclusive or selectable options among a set of status values.

In the status column, the static requirements are stated as follows:

- m For characteristics contained in mandatory packages or in conditional packages if the GDMO condition is always true.
- o For characteristics of conditional packages with GDMO conditions that indicate static optionality, e.g. “if an instance supports it”.
- cn For all other conditions, where “n” is a unique integer and “cn” is a reference to a conditional status expression as defined in ITU-T Rec. X.291 | ISO/IEC 9646-2 and ITU-T Rec. X.296 | ISO/IEC 9646-7. Each condition denoted by “cn” is relative to the containing table.
- x For characteristics explicitly prohibited by the definition.
- – For characteristics that are not mentioned by the definition.

The following common notations, defined in ITU-T Rec. X.724 | ISO/IEC 10165-6 and Rec. X.296 | ISO/IEC 9646-7 are used for the support answer columns:

- Y Implemented
- N Not implemented
- – No answer required

The following abbreviations are used:

smi2AttributeID	{ joint-iso-itu-t ms(9) smi(3) part2(2) attribute(7) }
smi2MObjectClass	{ joint-iso-itu-t ms(9) smi(3) part2(2) managedObjectClass(3) }
smi2Notification	{ joint-iso-itu-t ms(9) smi(3) part2(2) notification(10) }

H.1 Statement of conformance to the basicSpawnerClass object class

Table H.1 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	basicSpawnerClass	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx1(1)}		

If the answer to the actual class question in the managed object class support Table H.1 is no, the supplier of the implementation shall fill in the actual class support in Table H.2.

Table H.2 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.1.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the "Support" and "Additional information" columns in Table H.3.

Table H.3 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		m		
c1: if not (H-1/1b) then m else –						

H.1.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the "Support" and "Additional information" columns in Table H.4. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.4 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	allomorphs	{smi2AttributeID 50}		x		c1		x	
2	nameBinding	{smi2AttributeID 63}		–		m		x	
3	objectClass	{smi2AttributeID 65}		–		m		x	
4	packages	{smi2AttributeID 66}		–		m		x	

Table H.4 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	x		x		x		
c1: if not (H-1/1b) then m else –							

H.1.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.1.4 Actions

There are no actions defined for this object class.

H.1.5 Notifications

There are no notifications defined for this object class.

H.1.6 Parameters

There are no parameters defined for this object class.

H.2 Statement of conformance to the commandSequencer object class

Table H.5 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	commandSequencer	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx2(2)}		

If the answer to the actual class question in the managed object class support Table H.5 is no, the supplier of the implementation shall fill in the actual class support in Table H.6.

Table H.6 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.2.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the “Support” and “Additional information” columns in Table H.7.

Table H.7 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		c2		
c1: if not (H-5/1b) then m else – c2: if H-7/1 then m else –						

H.2.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.8. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.8 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	administrativeState	{smi2AttributeID 31}		m		m		m	
2	allomorphs	{smi2AttributeID 50}		x		c1		x	
3	commandSequencerId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx2(2)}		–		m		x	
4	nameBinding	{smi2AttributeID 63}		–		m		x	
5	objectClass	{smi2AttributeID 65}		–		m		x	
6	operationalState	{smi2AttributeID 35}		–		m		x	
7	packages	{smi2AttributeID 66}		–		c2		x	

Table H.8 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	x		x		x		
5	x		x		x		
6	x		x		x		
7	x		x		x		

c1: if not (H-5/1b) then m else –

c2: if H-7/2 then m else –

H.2.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.2.4 Actions

There are no actions defined for this object class.

H.2.5 Notifications

The supplier of the implementation shall state whether or not the notifications specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.9. The supplier of the implementation shall indicate support in terms of the confirmed and non-confirmed modes.

Table H.9 – MOCS – Notification support

Index	Notification type template label	Value of object identifier for notification type	Constraints and values	Status	Support		Additional information
					Confirmed	Non-confirmed	
1	objectCreation	{smi2Notification 6}		m			
2	objectDeletion	{smi2Notification 7}		m			
3	stateChange	{smi2Notification 14}		m			

Table H.9 *(continued)*

Index	Subindex	Notification field name label	Value of object identifier of attribute type associated with field	Constraints and values	Status	Support	Additional information
1	1.1	additionalInformation	{smi2AttributeID 6}		o		
	1.1.1	identifier	–		c:m		
	1.1.2	significance	–		c:m		
	1.1.3	information	–		c:m		
	1.2	additionalText	{smi2AttributeID 7}		o		
	1.3	attributeList	{smi2AttributeID 9}		o		
	1.3.1	attributeId	–		c:m		
	1.3.1.1	globalForm	–		c:o.1		
	1.3.1.2	localForm	–		c:o.1		
	1.3.2	attributeValue	–		c:m		
	1.4	correlatedNotifications	{smi2AttributeID 12}		o		
	1.4.1	correlatedNotifications	–		c:m		
	1.4.2	sourceObjectInst	–		c:o		
	1.4.2.1	distinguishedName	–		c:o.2		
	1.4.2.1.1	AttributeType	–		c:m		
	1.4.2.1.2	AttributeValue	–		c:m		
	1.4.2.2	nonSpecificForm	–		c:o.2		
	1.4.2.3	localDistinguishedName	–		c:o.2		
	1.4.2.3.1	AttributeType	–		c:m		
	1.4.2.3.2	AttributeValue	–		c:m		
	1.5	notificationIdentifier	{smi2AttributeID 16}		o		
	1.6	sourceIndicator	{smi2AttributeID 26}		o		
2	2.1	additionalInformation	{smi2AttributeID 6}		o		
	2.1.1	identifier	–		c:m		
	2.1.2	significance	–		c:m		
	2.1.3	information	–		c:m		
	2.2	additionalText	{smi2AttributeID 7}		o		
	2.3	attributeList	{smi2AttributeID 9}		o		
	2.3.1	attributeId	–		c:m		
	2.3.1.1	globalForm	–		c:o.3		
	2.3.1.2	localForm	–		c:o.3		
	2.3.2	attributeValue	–		c:m		
	2.4	correlatedNotifications	{smi2AttributeID 12}		o		
	2.4.1	correlatedNotifications	–		c:m		
	2.4.2	sourceObjectInst	–		c:o		
	2.4.2.1	distinguishedName	–		c:o.4		
	2.4.2.1.1	AttributeType	–		c:m		
	2.4.2.1.2	AttributeValue	–		c:m		
	2.4.2.2	nonSpecificForm	–		c:o.4		
	2.4.2.3	localDistinguishedName	–		c:o.4		

Table H.9 *(concluded)*

Index	Subindex	Notification field name label	Value of object identifier of attribute type associated with field	Constraints and values	Status	Support	Additional information
	2.4.2.3.1	AttributeType	–		c:m		
	2.4.2.3.2	AttributeValue	–		c:m		
	2.5	notificationIdentifier	{smi2AttributeID 16}		o		
	2.6	sourceIndicator	{smi2AttributeID 26}		o		
3	3.1	additionalInformation	{smi2AttributeID 6}		o		
	3.1.1	identifier	–		c:m		
	3.1.2	significance	–		c:m		
	3.1.3	information	–		c:m		
	3.2	additionalText	{smi2AttributeID 7}		o		
	3.3	attributeIdentifierList	{smi2AttributeID 8}		o		
	3.3.1	globalForm	–		c:o.5		
	3.3.2	localForm	–		c:o.5		
	3.4	correlatedNotifications	{smi2AttributeID 12}		o		
	3.4.1	correlatedNotifications	–		c:m		
	3.4.2	sourceObjectInst	–		c:o		
	3.4.2.1	distinguishedName	–		c:o.6		
	3.4.2.1.1	AttributeType	–		c:m		
	3.4.2.1.2	AttributeValue	–		c:m		
	3.4.2.2	nonSpecificForm	–		c:o.6		
	3.4.2.3	localDistinguishedName	–		c:o.6		
	3.4.2.3.1	AttributeType	–		c:m		
	3.4.2.3.2	AttributeValue	–		c:m		
	3.5	notificationIdentifier	{smi2AttributeID 16}		o		
	3.6	sourceIndicator	{smi2AttributeID 26}		o		
	3.7	stateChangeDefinition	{smi2AttributeID 28}		m		
	3.7.1	attributeID	–		m		
	3.7.1.1	globalForm	–		c:o.7		
	3.7.1.2	localForm	–		c:o.7		
	3.7.2	oldAttributeValue	–		o		
	3.7.3	newAttributeValue	–		m		

H.2.6 Parameters

There are no parameters defined for this object class.

H.3 Statement of conformance to the generalStringScript object class

Table H.10 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	generalStringScript	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx3(3)}		

If the answer to the actual class question in the managed object class support Table H.10 is no, the supplier of the implementation shall fill in the actual class support in Table H.11.

Table H.11 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.3.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the “Support” and “Additional information” columns in Table H.12.

Table H.12 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		m		
c1: if not (H-10/1b) then m else –						

H.3.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.13. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.13 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	administrativeState	{smi2AttributeID 31}		m		m		m	
2	allomorphs	{smi2AttributeID 50}		x		c1		x	
3	executionResultType	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx3(3)}		–		m		x	
4	nameBinding	{smi2AttributeID 63}		–		m		x	
5	objectClass	{smi2AttributeID 65}		–		m		x	
6	packages	{smi2AttributeID 66}		–		m		x	
7	scriptContent	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx4(4)}		m		m		m	
8	scriptId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx5(5)}		–		m		x	
9	scriptLanguageName	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx7(7)}		m		m		m	

Table H.13 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	x		x		x		
5	x		x		x		
6	x		x		x		
7	x		x		x		
8	x		x		x		
9	x		x		x		
c1: if not (H-10/1b) then m else –							

H.3.4 Attribute groups

There are no attribute groups defined for the managed object class.

H.3.5 Actions

There are no actions defined for this object class.

H.3.6 Notifications

There are no notifications defined for this object class.

H.3.7 Parameters

There are no parameters defined for this object class.

H.4 Statement of conformance to the launchPad object class

Table H.14 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	launchPad	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx6(6)}		

If the answer to the actual class question in the managed object class support Table H.14 is no, the supplier of the implementation shall fill in the actual class support in Table H.15.

Table H.15 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.4.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the “Support” and “Additional information” columns in Table H.16.

Table H.16 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		m		
c1: if not (H-14/1b) then m else –						

H.4.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.17. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.17 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	administrativeState	{smi2AttributeID 31}		–		x		x	
2	allomorphs	{smi2AttributeID 50}		x		c1		x	
3	availabilityStatus	{smi2AttributeID 33}		–		x		x	
4	controlStatus	{smi2AttributeID 34}		–		m		x	
5	launchPadId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx6(6)}		–		m		x	
6	nameBinding	{smi2AttributeID 63}		–		m		x	
7	objectClass	{smi2AttributeID 65}		–		m		x	
8	observedAttributeId	{joint-iso-itu-t ms(9) function(2) part11(11) attribute(7) xx15(15)}		m		m		m	
9	observedObjectInstance	{joint-iso-itu-t ms(9) function(2) part11(11) attribute(7) xx16(16)}		m		m		m	
10	operationalState	{smi2AttributeID 35}		–		x		x	
11	packages	{smi2AttributeID 66}		–		m		x	
12	schedulerName	{smi2AttributeID 67}		–		m		x	
13	usageState	{smi2AttributeID 39}		–		x		x	

Table H.17 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	x		x		x		
5	x		x		x		
6	x		x		x		
7	x		x		x		
8	x		x		x		
9	x		x		x		
10	x		x		x		
11	x		x		x		
12	x		x		x		
13	x		x		x		
c1: if not (H-14/1b) then m else –							

H.4.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.4.4 Actions

The supplier of the implementation shall state whether or not the actions specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.18.

Table H.18 – MOCS – Action support

Index	Action type template label	Value of object identifier for action type	Constraints and values	Status	Support	Additional information
1	resume	{joint-iso-itu-t ms(9) function(2) part21(21) action(9) xx1(1)}		m		
2	suspend	{joint-iso-itu-t ms(9) function(2) part21(21) action(9) xx2(2)}		m		
3	terminate	{joint-iso-itu-t ms(9) function(2) part21(21) action(9) xx3(3)}		m		

Table H.19 – MOCS – Action support

Index	Subindex	Action field name label	Constraints and values	Status	Support	Additional information
1	1.1	SpawnerObjectId		m		
	1.1.1	triggerId		m		
	1.1.1.1	distinguishedName		c:o.1		
	1.1.1.1.1	AttributeType		c:m		
	1.1.1.1.2	AttributeValue		c:m		
	1.1.1.2	nonSpecificForm		c:o.1		
	1.1.1.3	localDistinguishedName		c:o.1		
	1.1.1.3.1	AttributeType		c:m		
	1.1.1.3.2	AttributeValue		c:m		
	1.1.2	CHOICE		m		
	1.1.2.1	threadId		c:o.2		
	1.1.2.1.1	distinguishedName		c:o.3		
	1.1.2.1.1.1	AttributeType		c:m		
	1.1.2.1.1.2	AttributeValue		c:m		
	1.1.2.1.2	nonSpecificForm		c:o.3		
	1.1.2.1.3	localDistinguishedName		c:o.3		
	1.1.2.1.3.1	AttributeType		c:m		
	1.1.2.1.3.2	AttributeValue		c:m		
	1.1.2.2	launchPadId		c:o.2		
	1.1.2.2.1	distinguishedName		c:o.4		
	1.1.2.2.1.1	AttributeType		c:m		
	1.1.2.2.1.2	AttributeValue		c:m		
	1.1.2.2.2	nonSpecificForm		c:o.4		
	1.1.2.2.3	localDistinguishedName		c:o.4		
	1.1.2.2.3.1	AttributeType		c:m		
	1.1.2.2.3.2	AttributeValue		c:m		
2	2.1	SpawnerObjectId		m		
	2.1.1	triggerId		m		
	2.1.1.1	distinguishedName		c:o.5		
	2.1.1.1.1	AttributeType		c:m		
	2.1.1.1.2	AttributeValue		c:m		
	2.1.1.2	nonSpecificForm		c:o.5		
	2.1.1.3	localDistinguishedName		c:o.5		
	2.1.1.3.1	AttributeType		c:m		
	2.1.1.3.2	AttributeValue		c:m		
	2.1.2	CHOICE		m		
	2.1.2.1	threadId		c:o.6		
	2.1.2.1.1	distinguishedName		c:o.7		
	2.1.2.1.1.1	AttributeType		c:m		
	2.1.2.1.1.2	AttributeValue		c:m		
	2.1.2.1.2	nonSpecificForm		c:o.7		
	2.1.2.1.3	localDistinguishedName		c:o.7		
	2.1.2.1.3.1	AttributeType		c:m		
	2.1.2.1.3.2	AttributeValue		c:m		
	2.1.2.2	launchPadId		c:o.6		
	2.1.2.2.1	distinguishedName		c:o.8		
	2.1.2.2.1.1	AttributeType		c:m		
	2.1.2.2.1.2	AttributeValue		c:m		
	2.1.2.2.2	nonSpecificForm		c:o.8		
	2.1.2.2.3	localDistinguishedName		c:o.8		
	2.1.2.2.3.1	AttributeType		c:m		
	2.1.2.2.3.2	AttributeValue		c:m		
3	3.1	TriggerId		m		
	3.1.1	distinguishedName		c:o.9		
	3.1.1.1	AttributeType		c:m		
	3.1.1.2	AttributeValue		c:m		
	3.1.2	nonSpecificForm		c:o.9		
	3.1.3	localDistinguishedName		c:o.9		
	3.1.3.1	AttributeType		c:m		
	3.1.3.2	AttributeValue		c:m		

H.4.5 Notifications

There are no notifications defined for this object class.

H.4.6 Parameters

There are no parameters defined for this object class.

H.5 Statement of conformance to the asynchronousLaunchPad object class

Table H.20 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	asynchronousLaunchPad	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx4(4)}		

If the answer to the actual class question in the managed object class support Table H.20 is no, the supplier of the implementation shall fill in the actual class support in Table H.21.

Table H.21 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.5.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the “Support” and “Additional information” columns in Table H.22.

Table H.22 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		m		
c1: if not (H-20/1b) then m else –						

H.5.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.23. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

H.5.3 Attribute groups

There are no attribute groups defined for the managed object class.

Table H.23 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	administrativeState	{smi2AttributeID 31}		–		x		x	
2	allomorphs	{smi2AttributeID 50}		x		c1		x	
3	availabilityStatus	{smi2AttributeID 33}		–		x		x	
4	controlStatus	{smi2AttributeID 34}		–		m		x	
5	launchPadId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx6(6)}		–		m		x	
6	nameBinding	{smi2AttributeID 63}		–		m		x	
7	objectClass	{smi2AttributeID 65}		–		m		x	
8	observedAttributeId	{joint-iso-itu-t ms(9) function(2) part11(11) attribute(7) xx15(15)}		m		m		m	
9	observedObjectInstance	{joint-iso-itu-t ms(9) function(2) part11(11) attribute(7) xx16(16)}		m		m		m	
10	operationalState	{smi2AttributeID 35}		–		x		x	
11	packages	{smi2AttributeID 66}		–		m		x	
12	schedulerName	{smi2AttributeID 67}		–		m		x	
13	usageState	{smi2AttributeID 39}		–		x		x	

Table H.23 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	x		x		x		
5	x		x		x		
6	x		x		x		
7	x		x		x		
8	x		x		x		
9	x		x		x		
10	x		x		x		
11	x		x		x		
12	x		x		x		
13	x		x		x		
c1: if not (H-20/1b) then m else –							

H.5.4 Actions

The supplier of the implementation shall state whether or not the actions specified by all packages instantiated in a managed object of this class are supported, in the "Support" and "Additional information" columns in Table H.24.

Table H.24 – MOCS – Action support

Index	Action type template label	Value of object identifier for action type	Constraints and values	Status	Support	Additional information
1	resume	{joint-iso-itu-t ms(9) function(2) part21(21) action(9) xx1(1)}		m		
2	suspend	{joint-iso-itu-t ms(9) function(2) part21(21) action(9) xx2(2)}		m		
3	terminate	{joint-iso-itu-t ms(9) function(2) part21(21) action(9) xx3(3)}		m		

Table H.25 – MOCS – Action support

Index	Subindex	Action field name label	Constraints and values	Status	Support	Additional information
1	1.1	SpawnerObjectId		m		
	1.1.1	triggerId		m		
	1.1.1.1	distinguishedName		c:o.1		
	1.1.1.1.1	AttributeType		c:m		
	1.1.1.1.2	AttributeValue		c:m		
	1.1.1.2	nonSpecificForm		c:o.1		
	1.1.1.3	localDistinguishedName		c:o.1		
	1.1.1.3.1	AttributeType		c:m		
	1.1.1.3.2	AttributeValue		c:m		
	1.1.2	CHOICE		m		
	1.1.2.1	threadId		c:o.2		
	1.1.2.1.1	distinguishedName		c:o.3		
	1.1.2.1.1.1	AttributeType		c:m		
	1.1.2.1.1.2	AttributeValue		c:m		
	1.1.2.1.2	nonSpecificForm		c:o.3		
	1.1.2.1.3	localDistinguishedName		c:o.3		
	1.1.2.1.3.1	AttributeType		c:m		
	1.1.2.1.3.2	AttributeValue		c:m		
	1.1.2.2	launchPadId		c:o.2		
	1.1.2.2.1	distinguishedName		c:o.4		
	1.1.2.2.1.1	AttributeType		c:m		
	1.1.2.2.1.2	AttributeValue		c:m		
	1.1.2.2.2	nonSpecificForm		c:o.4		
	1.1.2.2.3	localDistinguishedName		c:o.4		
	1.1.2.2.3.1	AttributeType		c:m		
	1.1.2.2.3.2	AttributeValue		c:m		
2	2.1	SpawnerObjectId		m		
	2.1.1	triggerId		m		
	2.1.1.1	distinguishedName		c:o.5		
	2.1.1.1.1	AttributeType		c:m		
	2.1.1.1.2	AttributeValue		c:m		
	2.1.1.2	nonSpecificForm		c:o.5		

Table H.25 *(concluded)*

Index	Subindex	Action field name label	Constraints and values	Status	Support	Additional information
	2.1.1.3	localDistinguishedName		c:o.5		
	2.1.1.3.1	AttributeType		c:m		
	2.1.1.3.2	AttributeValue		c:m		
	2.1.2	CHOICE		m		
	2.1.2.1	threadId		c:o.6		
	2.1.2.1.1	distinguishedName		c:o.7		
	2.1.2.1.1.1	AttributeType		c:m		
	2.1.2.1.1.2	AttributeValue		c:m		
	2.1.2.1.2	nonSpecificForm		c:o.7		
	2.1.2.1.3	localDistinguishedName		c:o.7		
	2.1.2.1.3.1	AttributeType		c:m		
	2.1.2.1.3.2	AttributeValue		c:m		
	2.1.2.2	launchPadId		c:o.6		
	2.1.2.2.1	distinguishedName		c:o.8		
	2.1.2.2.1.1	AttributeType		c:m		
	2.1.2.2.1.2	AttributeValue		c:m		
	2.1.2.2.2	nonSpecificForm		c:o.8		
	2.1.2.2.3	localDistinguishedName		c:o.8		
	2.1.2.2.3.1	AttributeType		c:m		
	2.1.2.2.3.2	AttributeValue		c:m		
3	3.1	TriggerId		m		
	3.1.1	distinguishedName		c:o.9		
	3.1.1.1	AttributeType		c:m		
	3.1.1.2	AttributeValue		c:m		
	3.1.2	nonSpecificForm		c:o.9		
	3.1.3	localDistinguishedName		c:o.9		
	3.1.3.1	AttributeType		c:m		
	3.1.3.2	AttributeValue		c:m		

H.5.6 Notifications

There are no notifications defined for this object class.

H.5.7 Parameters

There are no parameters defined for this object class.

H.6 Statement of conformance to the synchronousLaunchPad object class

Table H.26 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	synchronousLaunchPad	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx5(5)}		

If the answer to the actual class question in the managed object class support Table H.26 is no, the supplier of the implementation shall fill in the actual class support in Table H.27.

Table H.27 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.6.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the "Support" and "Additional information" columns in Table H.28.

Table H.28 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		m		
c1: if not (H-26/1b) then m else –						

H.6.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the "Support" and "Additional information" columns in Table H.29. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.29 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	administrativeState	{smi2AttributeID 31}		–		x		x	
2	allomorphs	{smi2AttributeID 50}		x		c1		x	
3	availabilityStatus	{smi2AttributeID 33}		–		x		x	
4	controlStatus	{smi2AttributeID 34}		–		m		x	
5	launchPadId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx6(6)}		–		m		x	
6	nameBinding	{smi2AttributeID 63}		–		m		x	
7	objectClass	{smi2AttributeID 65}		–		m		x	
8	observedAttributeId	{joint-iso-itu-t ms(9) function(2) part11(11) attribute(7) xx15(15)}		m		m		m	
9	observedObjectInstance	{joint-iso-itu-t ms(9) function(2) part11(11) attribute(7) xx16(16)}		m		m		m	
10	operationalState	{smi2AttributeID 35}		–		x		x	
11	packages	{smi2AttributeID 66}		–		m		x	
12	schedulerName	{smi2AttributeID 67}		–		m		x	
13	usageState	{smi2AttributeID 39}		–		x		x	

Table H.29 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	x		x		x		
5	x		x		x		
6	x		x		x		
7	x		x		x		
8	x		x		x		
9	x		x		x		
10	x		x		x		
11	x		x		x		
12	x		x		x		
13	x		x		x		
c1: if not (H-26/1b) then m else –							

H.6.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.6.4 Actions

The supplier of the implementation shall state whether or not the actions specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.30.

Table H.30 – MOCS – Action support

Index	Action type template label	Value of object identifier for action type	Constraints and values	Status	Support	Additional information
1	resume	{joint-iso-itu-t ms(9) function(2) part21(21) action(9) xx1(1)}		m		
2	suspend	{joint-iso-itu-t ms(9) function(2) part21(21) action(9) xx2(2)}		m		
3	terminate	{joint-iso-itu-t ms(9) function(2) part21(21) action(9) xx3(3)}		m		

Table H.31 – MOCS – Action support

Index	Subindex	Action field name label	Constraints and values	Status	Support	Additional information
1	1.1	SpawnerObjectId		m		
	1.1.1	triggerId		m		
	1.1.1.1	distinguishedName		c:o.1		
	1.1.1.1.1	AttributeType		c:m		
	1.1.1.1.2	AttributeValue		c:m		
	1.1.1.2	nonSpecificForm		c:o.1		
	1.1.1.3	localDistinguishedName		c:o.1		
	1.1.1.3.1	AttributeType		c:m		
	1.1.1.3.2	AttributeValue		c:m		
	1.1.2	CHOICE		m		
	1.1.2.1	threadId		c:o.2		
	1.1.2.1.1	distinguishedName		c:o.3		
	1.1.2.1.1.1	AttributeType		c:m		
	1.1.2.1.1.2	AttributeValue		c:m		
	1.1.2.1.2	nonSpecificForm		c:o.3		
	1.1.2.1.3	localDistinguishedName		c:o.3		
	1.1.2.1.3.1	AttributeType		c:m		
	1.1.2.1.3.2	AttributeValue		c:m		
	1.1.2.2	launchPadId		c:o.2		
	1.1.2.2.1	distinguishedName		c:o.4		
	1.1.2.2.1.1	AttributeType		c:m		
	1.1.2.2.1.2	AttributeValue		c:m		
	1.1.2.2.2	nonSpecificForm		c:o.4		
	1.1.2.2.3	localDistinguishedName		c:o.4		
	1.1.2.2.3.1	AttributeType		c:m		
	1.1.2.2.3.2	AttributeValue		c:m		
2	2.1	SpawnerObjectId		m		
	2.1.1	triggerId		m		
	2.1.1.1	distinguishedName		c:o.5		
	2.1.1.1.1	AttributeType		c:m		
	2.1.1.1.2	AttributeValue		c:m		
	2.1.1.2	nonSpecificForm		c:o.5		
	2.1.1.3	localDistinguishedName		c:o.5		
	2.1.1.3.1	AttributeType		c:m		
	2.1.1.3.2	AttributeValue		c:m		
	2.1.2	CHOICE		m		
	2.1.2.1	threadId		c:o.6		
	2.1.2.1.1	distinguishedName		c:o.7		
	2.1.2.1.1.1	AttributeType		c:m		
	2.1.2.1.1.2	AttributeValue		c:m		
	2.1.2.1.2	nonSpecificForm		c:o.7		
	2.1.2.1.3	localDistinguishedName		c:o.7		
	2.1.2.1.3.1	AttributeType		c:m		
	2.1.2.1.3.2	AttributeValue		c:m		
	2.1.2.2	launchPadId		c:o.6		
	2.1.2.2.1	distinguishedName		c:o.8		
	2.1.2.2.1.1	AttributeType		c:m		
	2.1.2.2.1.2	AttributeValue		c:m		
	2.1.2.2.2	nonSpecificForm		c:o.8		
	2.1.2.2.3	localDistinguishedName		c:o.8		
	2.1.2.2.3.1	AttributeType		c:m		
	2.1.2.2.3.2	AttributeValue		c:m		
3	3.1	TriggerId		m		
	3.1.1	distinguishedName		c:o.9		
	3.1.1.1	AttributeType		c:m		
	3.1.1.2	AttributeValue		c:m		
	3.1.2	nonSpecificForm		c:o.9		
	3.1.3	localDistinguishedName		c:o.9		
	3.1.3.1	AttributeType		c:m		
	3.1.3.2	AttributeValue		c:m		

H.6.5 Notifications

There are no notifications defined for this object class.

H.6.6 Parameters

There are no parameters defined for this object class.

H.7 Statement of conformance to the launchScript object class

Table H.32 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	launchScript	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx7(7)}		

If the answer to the actual class question in the managed object class support Table H.32 is no, the supplier of the implementation shall fill in the actual class support in Table H.33.

Table H.33 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.7.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the “Support” and “Additional information” columns in Table H.34.

Table H.34 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		c2		
c1: if not (H-32/1b) then m else – c2: if H-34/1 then m else –						

H.7.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.35. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.35 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	administrativeState	{smi2AttributeID 31}		m		m		m	
2	allomorphs	{smi2AttributeID 50}		x		c1		x	
3	executionResultType	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx3(3)}		–		m		x	
4	nameBinding	{smi2AttributeID 63}		–		m		x	
5	objectClass	{smi2AttributeID 65}		–		m		x	
6	packages	{smi2AttributeID 66}		–		c2		x	
7	scriptId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx5(5)}		–		m		x	

Table H.35 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	x		x		x		
5	x		x		x		
6	x		x		x		
7	x		x		x		

c1: if not (H-32/1b) then m else –
c2: if H.34/2 then m else –

H.7.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.7.4 Actions

There are no actions defined for this object class.

H.7.5 Notifications

There are no notifications defined for this object class.

H.7.6 Parameters

There are no parameters defined for this object class.

H.8 Statement of conformance to the scriptReferencer object class

Table H.36 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	scriptReferencer	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx8(8)}		

If the answer to the actual class question in the managed object class support Table H.36 is no, the supplier of the implementation shall fill in the actual class support in Table H.37.

Table H.37 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.8.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the “Support” and “Additional information” columns in Table H.38.

Table H.38 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		m		
c1: if not (H-36/1b) then m else –						

H.8.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.39. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.39 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	allomorphs	{smi2AttributeID 50}		x		c1		x	
2	nameBinding	{smi2AttributeID 63}		–		m		x	
3	objectClass	{smi2AttributeID 65}		–		m		x	
4	packages	{smi2AttributeID 66}		–		m		x	

Table H.39 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	x		x		x		
c1: if not (H-36/1b) then m else –							

H.8.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.8.4 Actions

There are no actions defined for this object class.

H.8.5 Notifications

There are no notifications defined for this object class.

H.8.6 Parameters

There are no parameters defined for this object class.

H.9 Statement of conformance to the thread object class

Table H.40 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	thread	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx9(9)}		

If the answer to the actual class question in the managed object class support Table H.40 is no, the supplier of the implementation shall fill in the actual class support in Table H.41.

Table H.41 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.9.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the “Support” and “Additional information” columns in Table H.42.

Table H.42 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		m		
c1: if not (H-40/1b) then m else –						

H.9.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the "Support" and "Additional information" columns in Table H.43. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.43 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	allomorphs	{smi2AttributeID 50}		x		c1		x	
2	executingParameters	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx9(9)}		–		m		x	
3	nameBinding	{smi2AttributeID 63}		–		m		x	
4	objectClass	{smi2AttributeID 65}		–		m		x	
5	operationalState	{smi2AttributeID 35}		–		m		x	
6	packages	{smi2AttributeID 66}		–		m		x	
7	scriptId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx5(5)}		–		m		x	
8	threadId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx10(10)}		–		m		x	

Table H.43 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	x		x		x		
5	x		x		x		
6	x		x		x		
7	x		x		x		
8	x		x		x		
c1: if not (H-40/1b) then m else –							

H.9.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.9.4 Actions

There are no actions defined for this object class.

H.9.5 Notifications

The supplier of the implementation shall state whether or not the notifications specified by all packages instantiated in a managed object of this class are supported, in the "Support" and "Additional information" columns in Table H.44. The supplier of the implementation shall indicate support in terms of the confirmed and non-confirmed modes.

Table H.44 – MOCS – Notification support

					Support		
Index	Notification type template label	Value of object identifier for notification type	Constraints and values	Status	Confirmed	Non-confirmed	Additional information
1	processingErrorAlarm	{smi2Notification 10}		m			

Table H.44 *(continued)*

Index	Subindex	Notification field name label	Value of object identifier of attribute type associated with field	Constraints and values	Status	Support	Additional information
1	1.1	additionalInformation	{smi2AttributeID 6}		o		
	1.1.1	identifier	–		c:m		
	1.1.2	significance	–		c:m		
	1.1.3	information	–		c:m		
	1.2	additionalText	{smi2AttributeID 7}		o		
	1.3	backedUpStatus	{smi2AttributeID 11}		o		
	1.4	backUpObject	{smi2AttributeID 40}		o		
	1.4.1	objectName	–		c:o.1		
	1.4.1.1	distinguishedName	–		c:o.2		
	1.4.1.1.1	AttributeType	–		c:m		
	1.4.1.1.2	AttributeValue	–		c:m		
	1.4.1.2	nonSpecificForm	–		c:o.2		
	1.4.1.3	localDistinguishedName	–		c:o.2		
	1.4.1.3.1	AttributeType	–		c:m		
	1.4.1.3.2	AttributeValue	–		c:m		
	1.4.2	noObject	–		c:o.1		
	1.5	correlatedNotifications	{smi2AttributeID 12}		o		
	1.5.1	correlatedNotifications	–		c:m		
	1.5.2	sourceObjectInst	–		c:o		
	1.5.2.1	distinguishedName	–		c:o.3		
	1.5.2.1.1	AttributeType	–		c:m		
	1.5.2.1.2	AttributeValue	–		c:m		
	1.5.2.2	nonSpecificForm	–		c:o.3		
	1.5.2.3	localDistinguishedName	–		c:o.3		
	1.5.2.3.1	AttributeType	–		c:m		
	1.5.2.3.2	AttributeValue	–		c:m		
	1.6	monitoredAttributes	{smi2AttributeID 15}		o		
	1.6.1	attributeId	–		c:m		
	1.6.1.1	globalForm	–		c:o.4		
	1.6.1.2	localForm	–		c:o.4		
	1.6.2	attributeValue	–		c:m		
	1.7	notificationIdentifier	{smi2AttributeID 16}		o		

Table H.44 *(concluded)*

Index	Subindex	Notification field name label	Value of object identifier of attribute type associated with field	Constraints and values	Status	Support	Additional information
	1.8	perceivedSeverity	{smi2AttributeID 17}		m		
	1.9	probableCause	{smi2AttributeID 18}		m		
	1.9.1	globalValue	–		c:o.5		
	1.9.2	localValue	–		c:o.5		
	1.10	proposedRepairActions	{smi2AttributeID 19}		o		
	1.10.1	OBJECT IDENTIFIER	–		c:o.6		
	1.10.2	INTEGER	–		c:o.6		
	1.11	specificProblems	{smi2AttributeID 27}		o		
	1.11.1	OBJECT IDENTIFIER	–		c:o.7		
	1.11.2	INTEGER	–		c:o.7		
	1.12	stateChangeDefinition	{smi2AttributeID 28}		o		
	1.12.1	attributeID	–		c:m		
	1.12.1.1	globalForm	–		c:o.8		
	1.12.1.2	localForm	–		c:o.8		
	1.12.2	oldAttributeValue	–		c:o		
	1.12.3	newAttributeValue	–		c:m		
	1.13	thresholdInfo	{smi2AttributeID 29}		o		
	1.13.1	triggeredThreshold	–		c:m		
	1.13.1.1	globalForm	–		c:o.9		
	1.13.1.2	localForm	–		c:o.9		
	1.13.2	observedValue	–		c:m		
	1.13.2.1	integer	–		c:o.10		
	1.13.2.2	real	–		c:o.10		
	1.13.3	thresholdLevel	–		c:o		
	1.13.3.1	up	–		c:o.11		
	1.13.3.1.1	high	–		c:m		
	1.13.3.1.1.1	integer	–		c:o.12		
	1.13.3.1.1.2	real	–		c:o.12		
	1.13.3.1.2	low	–		c:o		
	1.13.3.1.2.1	integer	–		c:o.13		
	1.13.3.1.2.2	real	–		c:o.13		
	1.13.3.2	down	–		c:o.11		
	1.13.3.2.1	high	–		c:m		
	1.13.3.2.1.1	integer	–		c:o.14		
	1.13.3.2.1.2	real	–		c:o.14		
	1.13.3.2.2	low	–		c:m		
	1.13.3.2.2.1	integer	–		c:o.15		
	1.13.3.2.2.2	real	–		c:o.15		
	1.13.4	armTime	–		c:o		
	1.14	trendIndication	{smi2AttributeID 30}		o		

H.9.6 Parameters

There are no parameters defined for this object class.

H.10 Statement of conformance to the suspendableThread object class

Table H.45 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	suspendableThread	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx10(10)}		

If the answer to the actual class question in the managed object class support Table H.45 is no, the supplier of the implementation shall fill in the actual class support in Table H.46.

Table H.46 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.10.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the "Support" and "Additional information" columns in Table H.47.

Table H.47 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		m		
c1: if not (H-45/1b) then m else –						

H.10.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the "Support" and "Additional information" columns in Table H.48. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.48 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	allomorphs	{smi2AttributeID 50}		x		c1		x	
2	controlStatus	{smi2AttributeID 34}		–		m		x	
3	executingParameters	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx9(9)}		–		m		x	
4	nameBinding	{smi2AttributeID 63}		–		m		x	
5	objectClass	{smi2AttributeID 65}		–		m		x	
6	operationalState	{smi2AttributeID 35}		–		m		x	
7	packages	{smi2AttributeID 66}		–		m		x	
8	scriptId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx5(5)}		–		m		x	
9	threadId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx10(10)}		–		m		x	

Table H.48 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	x		x		x		
5	x		x		x		
6	x		x		x		
7	x		x		x		
8	x		x		x		
9	x		x		x		
c1: if not (H-45/1b) then m else –							

H.10.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.10.4 Actions

The supplier of the implementation shall state whether or not the actions specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.49.

Table H.49 – MOCS – Action support

Index	Action type template label	Value of object identifier for action type	Constraints and values	Status	Support	Additional information
1	resume	{joint-iso-itu-t ms(9) function(2) part21(21) action(9) xx1(1)}		m		
2	suspend	{joint-iso-itu-t ms(9) function(2) part21(21) action(9) xx2(2)}		m		

H.10.5 Notifications

The supplier of the implementation shall state whether or not the notifications specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.51. The supplier of the implementation shall indicate support in terms of the confirmed and non-confirmed modes.

Table H.50 – MOCS – Action support

Index	Subindex	Action field name label	Constraints and values	Status	Support	Additional information
1	1.1	SpawnerObjectId		m		
	1.1.1	triggerId		m		
	1.1.1.1	distinguishedName		c:o.1		
	1.1.1.1.1	AttributeType		c:m		
	1.1.1.1.2	AttributeValue		c:m		
	1.1.1.2	nonSpecificForm		c:o.1		
	1.1.1.3	localDistinguishedName		c:o.1		
	1.1.1.3.1	AttributeType		c:m		
	1.1.1.3.2	AttributeValue		c:m		
	1.1.2	CHOICE		m		
	1.1.2.1	threadId		c:o.2		
	1.1.2.1.1	distinguishedName		c:o.3		
	1.1.2.1.1.1	AttributeType		c:m		
	1.1.2.1.1.2	AttributeValue		c:m		
	1.1.2.1.2	nonSpecificForm		c:o.3		
	1.1.2.1.3	localDistinguishedName		c:o.3		
	1.1.2.1.3.1	AttributeType		c:m		
	1.1.2.1.3.2	AttributeValue		c:m		
	1.1.2.2	launchPadId		c:o.2		
	1.1.2.2.1	distinguishedName		c:o.4		
	1.1.2.2.1.1	AttributeType		c:m		
	1.1.2.2.1.2	AttributeValue		c:m		
	1.1.2.2.2	nonSpecificForm		c:o.4		
	1.1.2.2.3	localDistinguishedName		c:o.4		
	1.1.2.2.3.1	AttributeType		c:m		
	1.1.2.2.3.2	AttributeValue		c:m		
2	2.1	SpawnerObjectId		m		
	2.1.1	triggerId		m		
	2.1.1.1	distinguishedName		c:o.5		
	2.1.1.1.1	AttributeType		c:m		
	2.1.1.1.2	AttributeValue		c:m		
	2.1.1.2	nonSpecificForm		c:o.5		
	2.1.1.3	localDistinguishedName		c:o.5		
	2.1.1.3.1	AttributeType		c:m		
	2.1.1.3.2	AttributeValue		c:m		
	2.1.2	CHOICE		m		
	2.1.2.1	threadId		c:o.6		
	2.1.2.1.1	distinguishedName		c:o.7		
	2.1.2.1.1.1	AttributeType		c:m		
	2.1.2.1.1.2	AttributeValue		c:m		
	2.1.2.1.2	nonSpecificForm		c:o.7		
	2.1.2.1.3	localDistinguishedName		c:o.7		
	2.1.2.1.3.1	AttributeType		c:m		
	2.1.2.1.3.2	AttributeValue		c:m		
	2.1.2.2	launchPadId		c:o.6		
	2.1.2.2.1	distinguishedName		c:o.8		
	2.1.2.2.1.1	AttributeType		c:m		
	2.1.2.2.1.2	AttributeValue		c:m		
	2.1.2.2.2	nonSpecificForm		c:o.8		
	2.1.2.2.3	localDistinguishedName		c:o.8		
	2.1.2.2.3.1	AttributeType		c:m		
	2.1.2.2.3.2	AttributeValue		c:m		

Table H.51 – MOCS – Notification support

Index	Notification type template label	Value of object identifier for notification type	Constraints and values	Status	Support		Additional information
					Confirmed	Non-confirmed	
1	processingErrorAlarm	{smi2Notification 10}		m			

Table H.51 *(continued)*

Index	Subindex	Notification field name label	Value of object identifier of attribute type associated with field	Constraints and values	Status	Support	Additional information
1	1.1	additionalInformation	{smi2AttributeID 6}		o		
	1.1.1	identifier	–		c:m		
	1.1.2	significance	–		c:m		
	1.1.3	information	–		c:m		
	1.2	additionalText	{smi2AttributeID 7}		o		
	1.3	backedUpStatus	{smi2AttributeID 11}		o		
	1.4	backUpObject	{smi2AttributeID 40}		o		
	1.4.1	objectName	–		c:o.1		
	1.4.1.1	distinguishedName	–		c:o.2		
	1.4.1.1.1	AttributeType	–		c:m		
	1.4.1.1.2	AttributeValue	–		c:m		
	1.4.1.2	nonSpecificForm	–		c:o.2		
	1.4.1.3	localDistinguishedName	–		c:o.2		
	1.4.1.3.1	AttributeType	–		c:m		
	1.4.1.3.2	AttributeValue	–		c:m		
	1.4.2	noObject	–		c:o.1		
	1.5	correlatedNotifications	{smi2AttributeID 12}		o		
	1.5.1	correlatedNotifications	–		c:m		
	1.5.2	sourceObjectInst	–		c:o		
	1.5.2.1	distinguishedName	–		c:o.3		
	1.5.2.1.1	AttributeType	–		c:m		
	1.5.2.1.2	AttributeValue	–		c:m		
	1.5.2.2	nonSpecificForm	–		c:o.3		
	1.5.2.3	localDistinguishedName	–		c:o.3		
	1.5.2.3.1	AttributeType	–		c:m		
	1.5.2.3.2	AttributeValue	–		c:m		
	1.6	monitoredAttributes	{smi2AttributeID 15}		o		
	1.6.1	attributeId	–		c:m		
	1.6.1.1	globalForm	–		c:o.4		
	1.6.1.2	localForm	–		c:o.4		
	1.6.2	attributeValue	–		c:m		
	1.7	notificationIdentifier	{smi2AttributeID 16}		o		
	1.8	perceivedSeverity	{smi2AttributeID 17}		m		
	1.9	probableCause	{smi2AttributeID 18}		m		
	1.9.1	globalValue	–		c:o.5		
	1.9.2	localValue	–		c:o.5		
	1.10	proposedRepairActions	{smi2AttributeID 19}		o		
	1.10.1	OBJECT IDENTIFIER	–		c:o.6		
	1.10.2	INTEGER	–		c:o.6		
	1.11	specificProblems	{smi2AttributeID 27}		o		
	1.11.1	OBJECT IDENTIFIER	–		c:o.7		
	1.11.2	INTEGER	–		c:o.7		

Table H.51 *(concluded)*

Index	Subindex	Notification field name label	Value of object identifier of attribute type associated with field	Constraints and values	Status	Support	Additional information
	1.12	stateChangeDefinition	{smi2AttributeID 28}		o		
	1.12.1	attributeID	–		c:m		
	1.12.1.1	globalForm	–		c:o.8		
	1.12.1.2	localForm	–		c:o.8		
	1.12.2	oldAttributeValue	–		c:o		
	1.12.3	newAttributeValue	–		c:m		
	1.13	thresholdInfo	{smi2AttributeID 29}		o		
	1.13.1	triggeredThreshold	–		c:m		
	1.13.1.1	globalForm	–		c:o.9		
	1.13.1.2	localForm	–		c:o.9		
	1.13.2	observedValue	–		c:m		
	1.13.2.1	integer	–		c:o.10		
	1.13.2.2	real	–		c:o.10		
	1.13.3	thresholdLevel	–		c:o		
	1.13.3.1	up	–		c:o.11		
	1.13.3.1.1	high	–		c:m		
	1.13.3.1.1.1	integer	–		c:o.12		
	1.13.3.1.1.2	real	–		c:o.12		
	1.13.3.1.2	low	–		c:o		
	1.13.3.1.2.1	integer	–		c:o.13		
	1.13.3.1.2.2	real	–		c:o.13		
	1.13.3.2	down	–		c:o.11		
	1.13.3.2.1	high	–		c:m		
	1.13.3.2.1.1	integer	–		c:o.14		
	1.13.3.2.1.2	real	–		c:o.14		
	1.13.3.2.2	low	–		c:m		
	1.13.3.2.2.1	integer	–		c:o.15		
	1.13.3.2.2.2	real	–		c:o.15		
	1.13.4	armTime	–		c:o		
	1.14	trendIndication	{smi2AttributeID 30}		o		

H.10.6 Parameters

There are no parameters defined for this object class.

H.11 Statement of conformance to the eventDiscriminationCounter object class

Table H.52 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	eventDiscriminationCounter	{joint-iso-itu-t ms(9) ms(9) function(2) part21(21) managedObjectClass(3) xx11(11)}		

If the answer to the actual class question in the managed object class support Table H.52 is no, the supplier of the implementation shall fill in the actual class support in Table H.53.

Table H.53 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.11.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the "Support" and "Additional information" columns in Table H.54.

Table H.54 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	availabilityStatusPackage	{smi2Package 22}		c2		
3	counterAlarmPackage	{joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx15(15)}		c3		
4	dailyScheduling	{smi2Package 25}		o		
5	duration	{smi2Package 26}		c4		
6	externalScheduler	{smi2Package 27}		o		
7	packagesPackage	{smi2Package 16}		c5		
8	weeklyScheduling	{smi2Package 29}		o		

c1: if not (H-52/1b) then m else –

c2: if "any of the scheduling packages, (duration, weekly scheduling, external) are present" then m else –

c3: if "a counter is of finite size and a notification is triggered by a capacity alarm threshold" then m else –

c4: if "the discriminator function is scheduled to start at a specified time and stop at either a specified time or function continuously" then m else –

c5: if H-54/1 or H-54/2 or H-54/3 or H-54/4 or H-54/5 or H-54/6 or H-54/8 then m else –

H.11.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the "Support" and "Additional information" columns in Table H.55. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

H.11.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.11.4 Actions

There are no actions defined for this object class.

H.11.5 Notifications

The supplier of the implementation shall state whether or not the notifications specified by all packages instantiated in a managed object of this class are supported, in the "Support" and "Additional information" columns in Table H.56. The supplier of the implementation shall indicate support in terms of the confirmed and non-confirmed modes.

Table H.55 – MOCS – Attribute support

Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Set by create		Get		Replace	
				Status	Support	Status	Support	Status	Support
1	administrativeState	{smi2AttributeID 31}		m		m		m	
2	allomorphs	{smi2AttributeID 50}		x		c1		x	
3	availabilityStatus	{smi2AttributeID 33}		–		c2		x	
4	capacityAlarmThreshold	{smi2AttributeID 52}		c3		c3		c3	
5	counter	{smi2AttributeID 88}		–		m		x	
6	discriminatorConstruct	{smi2AttributeID 56}		m		m		m	
7	discriminatorId	{smi2AttributeID 1}		–		m		x	
8	intervalsOfDay	{smi2AttributeID 57}		o		o		o	
9	maxCounterSize	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx12(12)}		–		m		x	
10	nameBinding	{smi2AttributeID 63}		–		m		x	
11	objectClass	{smi2AttributeID 65}		–		m		x	
12	operationalState	{smi2AttributeID 35}		–		m		x	
13	packages	{smi2AttributeID 66}		–		c4		x	
14	schedulerName	{smi2AttributeID 67}		–		o		x	
15	startTime	{smi2AttributeID 68}		c5		c5		c5	
16	stopTime	{smi2AttributeID 69}		c5		c5		c5	
17	weekMask	{smi2AttributeID 71}		o		o		o	

Table H.55 *(concluded)*

Index	Add		Remove		Set to default		Additional information
	Status	Support	Status	Support	Status	Support	
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	c3		c3		x		
5	x		x		x		
6	x		x		m		
7	x		x		x		
8	o		o		o		
9	x		x		x		
10	x		x		x		
11	x		x		x		
12	x		x		x		
13	x		x		x		
14	x		x		x		
15	x		x		x		
16	x		x		c5		
17	o		o		o		

c1: if not (H-52/1b) then m else –
c2: if H-54/2 then m else –
c3: if H-54/3 then m else –
c4: if H-54/7 then m else –
c5: if H-54/5 then m else –

Table H.56 – MOCS – Notification support

					Support		
Index	Notification type template label	Value of object identifier for notification type	Constraints and values	Status	Confirmed	Non-confirmed	Additional information
1	attributeValueChange	{smi2Notification 1}		m			
2	objectCreation	{smi2Notification 6}		m			
3	objectDeletion	{smi2Notification 7}		m			
4	processingErrorAlarm	{smi2Notification 10}		m			
5	stateChange	{smi2Notification 14}		m			

Table H.56 *(continued)*

Index	Subindex	Notification field name label	Value of object identifier of attribute type associated with field	Constraints and values	Status	Support	Additional information
1	1.1	additionalInformation	{smi2AttributeID 6}		o		
	1.1.1	identifier	–		c:m		
	1.1.2	significance	–		c:m		
	1.1.3	information	–		c:m		
	1.2	additionalText	{smi2AttributeID 7}		o		
	1.3	attributeIdentifierList	{smi2AttributeID 8}		o		
	1.3.1	globalForm	–		c:o.1		
	1.3.2	localForm	–		c:o.1		
	1.4	attributeValueChangeDefinition	{smi2AttributeID 10}		m		
	1.4.1	attributeID	–		m		
	1.4.1.1	globalForm	–		c:o.2		
	1.4.1.2	localForm	–		c:o.2		
	1.4.2	oldAttributeValue	–		o		
	1.4.3	newAttributeValue	–		m		
	1.5	correlatedNotifications	{smi2AttributeID 12}		o		
	1.5.1	correlatedNotifications	–		c:m		
	1.5.2	sourceObjectInst	–		c:o		
	1.5.2.1	distinguishedName	–		c:o.3		
	1.5.2.1.1	AttributeType	–		c:m		
	1.5.2.1.2	AttributeValue	–		c:m		
	1.5.2.2	nonSpecificForm	–		c:o.3		
	1.5.2.3	localDistinguishedName	–		c:o.3		
	1.5.2.3.1	AttributeType	–		c:m		
	1.5.2.3.2	AttributeValue	–		c:m		
	1.6	notificationIdentifier	{smi2AttributeID 16}		o		
	1.7	sourceIndicator	{smi2AttributeID 26}		o		

Table H.56 *(continued)*

Index	Subindex	Notification field name label	Value of object identifier of attribute type associated with field	Constraints and values	Status	Support	Additional information
2	2.1	additionalInformation	{smi2AttributeID 6}		o		
	2.1.1	identifier	–		c:m		
	2.1.2	significance	–		c:m		
	2.1.3	information	–		c:m		
	2.2	additionalText	{smi2AttributeID 7}		o		
	2.3	attributeList	{smi2AttributeID 9}		o		
	2.3.1	attributeId	–		c:m		
	2.3.1.1	globalForm	–		c:o.4		
	2.3.1.2	localForm	–		c:o.4		
	2.3.2	attributeValue	–		c:m		
	2.4	correlatedNotifications	{smi2AttributeID 12}		o		
	2.4.1	correlatedNotifications	–		c:m		
	2.4.2	sourceObjectInst	–		c:o		
	2.4.2.1	distinguishedName	–		c:o.5		
	2.4.2.1.1	AttributeType	–		c:m		
	2.4.2.1.2	AttributeValue	–		c:m		
	2.4.2.2	nonSpecificForm	–		c:o.5		
	2.4.2.3	localDistinguishedName	–		c:o.5		
	2.4.2.3.1	AttributeType	–		c:m		
	2.4.2.3.2	AttributeValue	–		c:m		
	2.5	notificationIdentifier	{smi2AttributeID 16}		o		
	2.6	sourceIndicator	{smi2AttributeID 26}		o		
3	3.1	additionalInformation	{smi2AttributeID 6}		o		
	3.1.1	identifier	–		c:m		
	3.1.2	significance	–		c:m		
	3.1.3	information	–		c:m		
	3.2	additionalText	{smi2AttributeID 7}		o		
	3.3	attributeList	{smi2AttributeID 9}		o		
	3.3.1	attributeId	–		c:m		
	3.3.1.1	globalForm	–		c:o.6		
	3.3.1.2	localForm	–		c:o.6		
	3.3.2	attributeValue	–		c:m		
	3.4	correlatedNotifications	{smi2AttributeID 12}		o		
	3.4.1	correlatedNotifications	–		c:m		
	3.4.2	sourceObjectInst	–		c:o		
	3.4.2.1	distinguishedName	–		c:o.7		
	3.4.2.1.1	AttributeType	–		c:m		
	3.4.2.1.2	AttributeValue	–		c:m		
	3.4.2.2	nonSpecificForm	–		c:o.7		
	3.4.2.3	localDistinguishedName	–		c:o.7		
	3.4.2.3.1	AttributeType	–		c:m		
	3.4.2.3.2	AttributeValue	–		c:m		
	3.5	notificationIdentifier	{smi2AttributeID 16}		o		
	3.6	sourceIndicator	{smi2AttributeID 26}		o		
4	4.1	additionalInformation	{smi2AttributeID 6}		o		
	4.1.1	identifier	–		c:m		
	4.1.2	significance	–		c:m		
	4.1.3	information	–		c:m		
	4.2	additionalText	{smi2AttributeID 7}		o		
	4.3	backedUpStatus	{smi2AttributeID 11}		o		
	4.4	backUpObject	{smi2AttributeID 40}		o		
	4.4.1	objectName	–		c:o.8		
	4.4.1.1	distinguishedName	–		c:o.9		
	4.4.1.1.1	AttributeType	–		c:m		

Table H.56 *(continued)*

Index	Subindex	Notification field name label	Value of object identifier of attribute type associated with field	Constraints and values	Status	Support	Additional information
	4.4.1.1.2	AttributeValue	–		c:m		
	4.4.1.2	nonSpecificForm	–		c:o.9		
	4.4.1.3	localDistinguishedName	–		c:o.9		
	4.4.1.3.1	AttributeType	–		c:m		
	4.4.1.3.2	AttributeValue	–		c:m		
	4.4.2	noObject	–		c:o.8		
	4.5	correlatedNotifications	{smi2AttributeID 12}		o		
	4.5.1	correlatedNotifications	–		c:m		
	4.5.2	sourceObjectInst	–		c:o		
	4.5.2.1	distinguishedName	–		c:o.10		
	4.5.2.1.1	AttributeType	–		c:m		
	4.5.2.1.2	AttributeValue	–		c:m		
	4.5.2.2	nonSpecificForm	–		c:o.10		
	4.5.2.3	localDistinguishedName	–		c:o.10		
	4.5.2.3.1	AttributeType	–		c:m		
	4.5.2.3.2	AttributeValue	–		c:m		
	4.6	monitoredAttributes	{smi2AttributeID 15}		o		
	4.6.1	attributeId	–		c:m		
	4.6.1.1	globalForm	–		c:o.11		
	4.6.1.2	localForm	–		c:o.11		
	4.6.2	attributeValue	–		c:m		
	4.7	notificationIdentifier	{smi2AttributeID 16}		o		
	4.8	perceivedSeverity	{smi2AttributeID 17}		m		
	4.9	probableCause	{smi2AttributeID 18}		m		
	4.9.1	globalValue	–		c:o.12		
	4.9.2	localValue	–		c:o.12		
	4.10	proposedRepairActions	{smi2AttributeID 19}		o		
	4.10.1	OBJECT IDENTIFIER	–		c:o.13		
	4.10.2	INTEGER	–		c:o.13		
	4.11	specificProblems	{smi2AttributeID 27}		o		
	4.11.1	OBJECT IDENTIFIER	–		c:o.14		
	4.11.2	INTEGER	–		c:o.14		
	4.12	stateChangeDefinition	{smi2AttributeID 28}		o		
	4.12.1	attributeID	–		c:m		
	4.12.1.1	globalForm	–		c:o.15		
	4.12.1.2	localForm	–		c:o.15		
	4.12.2	oldAttributeValue	–		c:o		
	4.12.3	newAttributeValue	–		c:m		
	4.13	thresholdInfo	{smi2AttributeID 29}		o		
	4.13.1	triggeredThreshold	–		c:m		
	4.13.1.1	globalForm	–		c:o.16		
	4.13.1.2	localForm	–		c:o.16		
	4.13.2	observedValue	–		c:m		
	4.13.2.1	integer	–		c:o.17		
	4.13.2.2	real	–		c:o.17		
	4.13.3	thresholdLevel	–		c:o		
	4.13.3.1	up	–		c:o.18		
	4.13.3.1.1	high	–		c:m		
	4.13.3.1.1.1	integer	–		c:o.19		
	4.13.3.1.1.2	real	–		c:o.19		
	4.13.3.1.2	low	–		c:o		
	4.13.3.1.2.1	integer	–		c:o.20		
	4.13.3.1.2.2	real	–		c:o.20		

Table H.56 *(concluded)*

Index	Subindex	Notification field name label	Value of object identifier of attribute type associated with field	Constraints and values	Status	Support	Additional information
	4.13.3.2	down	–		c:o.18		
	4.13.3.2.1	high	–		c:m		
	4.13.3.2.1.1	integer	–		c:o.21		
	4.13.3.2.1.2	real	–		c:o.21		
	4.13.3.2.2	low	–		c:m		
	4.13.3.2.2.1	integer	–		c:o.22		
	4.13.3.2.2.2	real	–		c:o.22		
	4.13.4	armTime	–		c:o		
	4.14	trendIndication	{smi2AttributeID 30}		o		
5	5.1	additionalInformation	{smi2AttributeID 6}		o		
	5.1.1	identifier	–		c:m		
	5.1.2	significance	–		c:m		
	5.1.3	information	–		c:m		
	5.2	additionalText	{smi2AttributeID 7}		o		
	5.3	attributeIdentifierList	{smi2AttributeID 8}		o		
	5.3.1	globalForm	–		c:o.23		
	5.3.2	localForm	–		c:o.23		
	5.4	correlatedNotifications	{smi2AttributeID 12}		o		
	5.4.1	correlatedNotifications	–		c:m		
	5.4.2	sourceObjectInst	–		c:o		
	5.4.2.1	distinguishedName	–		c:o.24		
	5.4.2.1.1	AttributeType	–		c:m		
	5.4.2.1.2	AttributeValue	–		c:m		
	5.4.2.2	nonSpecificForm	–		c:o.24		
	5.4.2.3	localDistinguishedName	–		c:o.24		
	5.4.2.3.1	AttributeType	–		c:m		
	5.4.2.3.2	AttributeValue	–		c:m		
	5.5	notificationIdentifier	{smi2AttributeID 16}		o		
	5.6	sourceIndicator	{smi2AttributeID 26}		o		
	5.7	stateChangeDefinition	{smi2AttributeID 28}		m		
	5.7.1	attributeID	–		m		
	5.7.1.1	globalForm	–		c:o.25		
	5.7.1.2	localForm	–		c:o.25		
	5.7.2	oldAttributeValue	–		o		
	5.7.3	newAttributeValue	–		m		

H.11.6 Parameters

There are no parameters defined for this object class.

H.12 Statement of conformance to the cmipCS object class

Table H.57 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	cmipCS	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx18(18)}		

If the answer to the actual class question in the managed object class support Table H.57 is no, the supplier of the implementation shall fill in the actual class support in Table H.58.

Table H.58 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.12.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the "Support" and "Additional information" columns in Table H.59.

Table H.59 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		m		
c1: if not (H-57/1b) then m else –						

H.12.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the "Support" and "Additional information" columns in Table H.60. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.60 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	administrativeState	{smi2AttributeID 31}		m		m		m	
2	aetitle	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx20(20)}		–		m		x	
3	allomorphs	{smi2AttributeID 50}		x		c1		x	
4	commandSequencerId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx2(2)}		–		m		x	
5	nameBinding	{smi2AttributeID 63}		–		m		x	
6	objectClass	{smi2AttributeID 65}		–		m		x	
7	operationalState	{smi2AttributeID 35}		–		m		x	
8	packages	{smi2AttributeID 66}		–		m		x	

Table H.60 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	x		x		x		
5	x		x		x		
6	x		x		x		
7	x		x		x		
8	x		x		x		
c1: if not (H-57/1b) then m else –							

H.12.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.12.4 Actions

There are no actions defined for this object class.

H.12.5 Notifications

The supplier of the implementation shall state whether or not the notifications specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.61. The supplier of the implementation shall indicate support in terms of the confirmed and non-confirmed modes.

Table H.61 – MOCS – Notification support

					Support		
Index	Notification type template label	Value of object identifier for notification type	Constraints and values	Status	Confirmed	Non-confirmed	Additional information
1	objectCreation	{smi2Notification 6}		m			
2	objectDeletion	{smi2Notification 7}		m			
3	stateChange	{smi2Notification 14}		m			

Table H.61 *(continued)*

Index	Subindex	Notification field name label	Value of object identifier of attribute type associated with field	Constraints and values	Status	Support	Additional information
1	1.1	additionalInformation	{smi2AttributeID 6}		o		
	1.1.1	identifier	–		c:m		
	1.1.2	significance	–		c:m		
	1.1.3	information	–		c:m		
	1.2	additionalText	{smi2AttributeID 7}		o		
	1.3	attributeList	{smi2AttributeID 9}		o		
	1.3.1	attributeId	–		c:m		
	1.3.1.1	globalForm	–		c:o.1		
	1.3.1.2	localForm	–		c:o.1		
	1.3.2	attributeValue	–		c:m		
	1.4	correlatedNotifications	{smi2AttributeID 12}		o		
	1.4.1	correlatedNotifications	–		c:m		
	1.4.2	sourceObjectInst	–		c:o		
	1.4.2.1	distinguishedName	–		c:o.2		
	1.4.2.1.1	AttributeType	–		c:m		
	1.4.2.1.2	AttributeValue	–		c:m		
	1.4.2.2	nonSpecificForm	–		c:o.2		

Table H.61 *(concluded)*

Index	Subindex	Notification field name label	Value of object identifier of attribute type associated with field	Constraints and values	Status	Support	Additional information
	1.4.2.3	localDistinguishedName	–		c:o.2		
	1.4.2.3.1	AttributeType	–		c:m		
	1.4.2.3.2	AttributeValue	–		c:m		
	1.5	notificationIdentifier	{smi2AttributeID 16}		o		
	1.6	sourceIndicator	{smi2AttributeID 26}		o		
2	2.1	additionalInformation	{smi2AttributeID 6}		o		
	2.1.1	identifier	–		c:m		
	2.1.2	significance	–		c:m		
	2.1.3	information	–		c:m		
	2.2	additionalText	{smi2AttributeID 7}		o		
	2.3	attributeList	{smi2AttributeID 9}		o		
	2.3.1	attributeId	–		c:m		
	2.3.1.1	globalForm	–		c:o.3		
	2.3.1.2	localForm	–		c:o.3		
	2.3.2	attributeValue	–		c:m		
	2.4	correlatedNotifications	{smi2AttributeID 12}		o		
	2.4.1	correlatedNotifications	–		c:m		
	2.4.2	sourceObjectInst	–		c:o		
	2.4.2.1	distinguishedName	–		c:o.4		
	2.4.2.1.1	AttributeType	–		c:m		
	2.4.2.1.2	AttributeValue	–		c:m		
	2.4.2.2	nonSpecificForm	–		c:o.4		
	2.4.2.3	localDistinguishedName	–		c:o.4		
	2.4.2.3.1	AttributeType	–		c:m		
	2.4.2.3.2	AttributeValue	–		c:m		
	2.5	notificationIdentifier	{smi2AttributeID 16}		o		
	2.6	sourceIndicator	{smi2AttributeID 26}		o		
3	3.1	additionalInformation	{smi2AttributeID 6}		o		
	3.1.1	identifier	–		c:m		
	3.1.2	significance	–		c:m		
	3.1.3	information	–		c:m		
	3.2	additionalText	{smi2AttributeID 7}		o		
	3.3	attributeIdentifierList	{smi2AttributeID 8}		o		
	3.3.1	globalForm	–		c:o.5		
	3.3.2	localForm	–		c:o.5		
	3.4	correlatedNotifications	{smi2AttributeID 12}		o		
	3.4.1	correlatedNotifications	–		c:m		
	3.4.2	sourceObjectInst	–		c:o		
	3.4.2.1	distinguishedName	–		c:o.6		
	3.4.2.1.1	AttributeType	–		c:m		
	3.4.2.1.2	AttributeValue	–		c:m		
	3.4.2.2	nonSpecificForm	–		c:o.6		
	3.4.2.3	localDistinguishedName	–		c:o.6		
	3.4.2.3.1	AttributeType	–		c:m		
	3.4.2.3.2	AttributeValue	–		c:m		
	3.5	notificationIdentifier	{smi2AttributeID 16}		o		
	3.6	sourceIndicator	{smi2AttributeID 26}		o		
	3.7	stateChangeDefinition	{smi2AttributeID 28}		m		
	3.7.1	attributeID	–		m		
	3.7.1.1	globalForm	–		c:o.7		
	3.7.1.2	localForm	–		c:o.7		
	3.7.2	oldAttributeValue	–		o		
	3.7.3	newAttributeValue	–		m		

H.12.6 Parameters

There are no parameters defined for this object class.

H.13 Statement of conformance to the cmisScript object class

Table H.62 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	cmisScript	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx12(12)}		

If the answer to the actual class question in the managed object class support Table H.62 is no, the supplier of the implementation shall fill in the actual class support in Table H.63.

Table H.63 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.13.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the “Support” and “Additional information” columns in Table H.64.

Table H.64 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		c2		
c1: if not (H-62/1b) then m else – c2: if H-64/1 then m else –						

H.13.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.65. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.65 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	administrativeState	{smi2AttributeID 31}		m		m		m	
2	allomorphs	{smi2AttributeID 50}		x		c1		x	
3	executionResultType	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx3(3)}		–		m		x	
4	nameBinding	{smi2AttributeID 63}		–		m		x	
5	objectClass	{smi2AttributeID 65}		–		m		x	
6	packages	{smi2AttributeID 66}		–		c2		x	
7	scriptId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx5(5)}		–		m		x	

Table H.65 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	x		x		x		
5	x		x		x		
6	x		x		x		
7	x		x		x		
c1: if not(H-62/1b) then m else – c2: if H-64/2 then m else –							

H.13.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.13.4 Actions

There are no actions defined for this object class.

H.13.5 Notifications

There are no notifications defined for this object class.

H.13.6 Parameters

There are no parameters defined for this object class.

H.14 Statement of conformance to the getCmisScript object class

Table H.66 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	getCmisScript	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx13(13)}		

If the answer to the actual class question in the managed object class support Table H.66 is no, the supplier of the implementation shall fill in the actual class support in Table H.67.

Table H.67 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.14.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the “Support” and “Additional information” columns in Table H.68.

Table H.68 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		c2		
c1: if not (H-66/1b) then m else – c2: if H-68/1 then m else –						

H.14.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.69. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.69 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	administrativeState	{smi2AttributeID 31}		m		m		m	
2	allomorphs	{smi2AttributeID 50}		x		c1		x	
3	attributeIdentifierList	{smi2AttributeID 8}		m		m		m	
4	baseManagedObjectId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx13(13)}		–		m		x	
5	executionResultType	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx3(3)}		–		m		x	
6	filter	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx15(15)}		m		m		m	
7	nameBinding	{smi2AttributeID 63}		–		m		x	
8	objectClass	{smi2AttributeID 65}		–		m		x	
9	packages	{smi2AttributeID 66}		–		c2		x	
10	scope	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx14(14)}		m		m		m	
11	scriptId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx5(5)}		–		m		x	
12	synchronization	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx16(16)}		m		m		m	

Table H.69 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	m		m		x		
4	x		x		x		
5	x		x		x		
6	x		x		x		
7	x		x		x		
8	x		x		x		
9	x		x		x		
10	x		x		x		
11	x		x		x		
12	x		x		x		
c1: if not (H-66/1b) then m else – c2: if H-68/2 then m else –							

H.14.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.14.4 Actions

There are no actions defined for this object class.

H.14.5 Notifications

There are no notifications defined for this object class.

H.14.6 Parameters

There are no parameters defined for this object class.

H.15 Statement of conformance to the setCmisScript object class

Table H.70 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	setCmisScript	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx14(14)}		

If the answer to the actual class question in the managed object class support Table H.70 is no, the supplier of the implementation shall fill in the actual class support in Table H.71.

Table H.71 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.15.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the "Support" and "Additional information" columns in Table H.72.

Table H.72 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		c2		
c1: if not (H-70/1b) then m else – c2: if H-72/1 then m else –						

H.15.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the "Support" and "Additional information" columns in Table H.73. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.73 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	administrativeState	{smi2AttributeID 31}		m		m		m	
2	allomorphs	{smi2AttributeID 50}		x		c1		x	
3	baseManagedObjectId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx13(13)}		–		m		x	
4	executionResultType	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx3(3)}		–		m		x	
5	filter	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx15(15)}		m		m		m	
6	modificationList	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx17(17)}		m		m		m	
7	nameBinding	{smi2AttributeID 63}		–		m		x	
8	objectClass	{smi2AttributeID 65}		–		m		x	
9	packages	{smi2AttributeID 66}		–		c2		x	
10	scope	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx14(14)}		m		m		m	
11	scriptId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx5(5)}		–		m		x	
12	synchronization	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx16(16)}		m		m		m	

Table H.73 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	x		x		x		
5	x		x		x		
6	m		m		x		
7	x		x		x		
8	x		x		x		
9	x		x		x		
10	x		x		x		
11	x		x		x		
12	x		x		x		
c1: if not (H-70/1b) then m else – c2: if H-72/2 then m else –							

H.15.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.15.4 Actions

There are no actions defined for this object class.

H.15.5 Notifications

There are no notifications defined for this object class.

H.15.6 Parameters

There are no parameters defined for this object class.

H.16 Statement of conformance to the actionCmisScript object class

Table H.74 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	actionCmisScript	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx15(15)}		

If the answer to the actual class question in the managed object class support Table H.74 is no, the supplier of the implementation shall fill in the actual class support in Table H.75.

Table H.75 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.16.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the "Support" and "Additional information" columns in Table H.76.

Table H.76 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		c2		
c1: if not (H-74/1b) then m else – c2: if H-76/1 then m else –						

H.16.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the "Support" and "Additional information" columns in Table H.77. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.77 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	administrativeState	{smi2AttributeID 31}		m		m		m	
2	allomorphs	{smi2AttributeID 50}		x		c1		x	
3	baseManagedObjectId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx13(13)}		–		m		x	
4	executionResultType	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx3(3)}		–		m		x	
5	filter	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx15(15)}		m		m		m	
6	nameBinding	{smi2AttributeID 63}		–		m		x	
7	objectClass	{smi2AttributeID 65}		–		m		x	
8	packages	{smi2AttributeID 66}		–		c2		x	
9	scope	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx14(14)}		m		m		m	
10	scriptId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx5(5)}		–		m		x	
11	synchronization	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx16(16)}		m		m		m	

Table H.77 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	x		x		x		
5	x		x		x		
6	x		x		x		
7	x		x		x		
8	x		x		x		
9	x		x		x		
10	x		x		x		
11	x		x		x		
c1: if not (H-74/1b) then m else – c2: if H-76/2 then m else –							

H.16.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.16.4 Actions

There are no actions defined for this object class.

H.16.5 Notifications

There are no notifications defined for this object class.

H.16.6 Parameters

There are no parameters defined for this object class.

H.17 Statement of conformance to the createCmisScript object class

Table H.78 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	createCmisScript	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx16(16)}		

If the answer to the actual class question in the managed object class support Table H.78 is no, the supplier of the implementation shall fill in the actual class support in Table H.79.

Table H.79 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.17.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the “Support” and “Additional information” columns in Table H.80.

Table H.80 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	managedObjectInstancePackage	{joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx16(16)}		c2		
3	packagesPackage	{smi2Package 16}		c3		
4	referenceObjectInstancePackage	{joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx18(18)}		c4		
5	superiourObjectInstancePackage	{joint-iso-itu-t ms(9) function(2) part21(21) package(4) xx17(17)}		c5		

c1: if not (H-78/1b) then m else –

c2: if “the superiourObjectInstancePackage is not present” then m else –

c3: if H-80/1 or H-80/2 or H-80/4 or H-80/5 then m else –

c4: if “the manager has the specified value” then m else –

c5: if “the managedObjectInstance Package is not present” then m else –

H.17.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.81. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.81 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	administrativeState	{smi2AttributeID 31}		m		m		m	
2	allomorphs	{smi2AttributeID 50}		x		c1		x	
3	attributeList	{smi2AttributeID 9}		m		m		m	
4	executionResultType	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx3(3)}		–		m		x	
5	managedObjectInstance	{smi2AttributeID 61}		c2		c2		c2	
6	nameBinding	{smi2AttributeID 63}		–		m		x	
7	objectClass	{smi2AttributeID 65}		–		m		x	
8	packages	{smi2AttributeID 66}		–		c3		x	
9	referenceObjectInstance	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx19(19)}		c4		c4		c4	
10	scriptId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx5(5)}		–		m		x	
11	superiourObjectInstance	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx18(18)}		c5		c5		c5	

Table H.81 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	m		m		x		
4	x		x		x		
5	x		x		x		
6	x		x		x		
7	x		x		x		
8	x		x		x		
9	x		x		x		
10	x		x		x		
11	x		x		x		

c1: if not (H-78/1b) then m else –
c2: if H-80/2 then m else –
c3: if H-80/3 then m else –
c4: if H-80/4 then m else –
c5: if H-80/5 then m else –

H.17.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.17.4 Actions

There are no actions defined for this object class.

H.17.5 Notifications

There are no notifications defined for this object class.

H.17.6 Parameters

There are no parameters defined for this object class.

H.18 Statement of conformance to the deleteCmisScript object class

Table H.82 – MOCS – Managed object class support

Index	Managed object class template label	Value of object identifier for class	Support of all mandatory features	Is the actual class the same as the managed object class to which conformance is claimed? (Y/N)
1	deleteCmisScript	{joint-iso-itu-t ms(9) function(2) part21(21) managedObjectClass(3) xx17(17)}		

If the answer to the actual class question in the managed object class support Table H.82 is no, the supplier of the implementation shall fill in the actual class support in Table H.83.

Table H.83 – MOCS – Actual class support

Index	Actual managed object class template label	Value of object identifier for actual class	Additional information
1			
2			

H.18.1 Packages

The supplier of the implementation shall state whether or not the conditional packages specified by this class are supported by an instance of this class, in the “Support” and “Additional information” columns in Table H.84.

Table H.84 – MOCS – Package support

Index	Package template label	Value of object identifier for package	Constraints and values	Status	Support	Additional information
1	allomorphicPackage	{smi2Package 17}		c1		
2	packagesPackage	{smi2Package 16}		c2		
c1: if not (H-82/1b) then m else – c2: if H-84/1 then m else –						

H.18.2 Attributes

The supplier of the implementation shall state whether or not the attributes specified by all packages instantiated in a managed object of this class are supported, in the “Support” and “Additional information” columns in Table H.85. The supplier of the implementation shall indicate support for each of the operations for each attribute supported.

Table H.85 – MOCS – Attribute support

				Set by create		Get		Replace	
Index	Attribute template label	Value of object identifier for attribute	Constraints and values	Status	Support	Status	Support	Status	Support
1	administrativeState	{smi2AttributeID 31}		m		m		m	
2	allomorphs	{smi2AttributeID 50}		x		c1		x	
3	baseManagedObjectId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx13(13)}		–		m		x	
4	executionResultType	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx3(3)}		–		m		x	
5	filter	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx15(15)}		m		m		m	
6	nameBinding	{smi2AttributeID 63}		–		m		x	
7	objectClass	{smi2AttributeID 65}		–		m		x	
8	packages	{smi2AttributeID 66}		–		c2		x	
9	scope	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx14(14)}		m		m		m	
10	scriptId	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx5(5)}		–		m		x	
11	synchronization	{joint-iso-itu-t ms(9) function(2) part21(21) attribute(7) xx16(16)}		m		m		m	

Table H.85 *(concluded)*

	Add		Remove		Set to default		
Index	Status	Support	Status	Support	Status	Support	Additional information
1	x		x		x		
2	x		x		x		
3	x		x		x		
4	x		x		x		
5	x		x		x		
6	x		x		x		
7	x		x		x		
8	x		x		x		
9	x		x		x		
10	x		x		x		
11	x		x		x		
c1: if not (H-82/1b) then m else – c2: if H-84/2 then m else –							

H.18.3 Attribute groups

There are no attribute groups defined for the managed object class.

H.18.4 Actions

There are no actions defined for this object class.

H.18.5 Notifications

There are no notifications defined for this object class.

H.18.6 Parameters

There are no parameters defined for this object class.

SERIES DE RECOMENDACIONES DEL UIT-T

Serie A	Organización del trabajo del UIT-T
Serie B	Medios de expresión: definiciones, símbolos, clasificación
Serie C	Estadísticas generales de telecomunicaciones
Serie D	Principios generales de tarificación
Serie E	Explotación general de la red, servicio telefónico, explotación del servicio y factores humanos
Serie F	Servicios de telecomunicación no telefónicos
Serie G	Sistemas y medios de transmisión, sistemas y redes digitales
Serie H	Sistemas audiovisuales y multimedios
Serie I	Red digital de servicios integrados
Serie J	Transmisiones de señales radiofónicas, de televisión y de otras señales multimedios
Serie K	Protección contra las interferencias
Serie L	Construcción, instalación y protección de los cables y otros elementos de planta exterior
Serie M	RGT y mantenimiento de redes: sistemas de transmisión, circuitos telefónicos, telegrafía, facsímil y circuitos arrendados internacionales
Serie N	Mantenimiento: circuitos internacionales para transmisiones radiofónicas y de televisión
Serie O	Especificaciones de los aparatos de medida
Serie P	Calidad de transmisión telefónica, instalaciones telefónicas y redes locales
Serie Q	Conmutación y señalización
Serie R	Transmisión telegráfica
Serie S	Equipos terminales para servicios de telegrafía
Serie T	Terminales para servicios de telemática
Serie U	Conmutación telegráfica
Serie V	Comunicación de datos por la red telefónica
Serie X	**Redes de datos y comunicación entre sistemas abiertos**
Serie Y	Infraestructura mundial de la información
Serie Z	Lenguajes de programación