INTERNATIONAL STANDARD

ISO/IEC 9506-3

First edition
1991-08-01

Industrial automation systems — Manufacturing message specification —

Part 3:
Companion standard for robotics

Systèmes d'automatisation industrielle — Système de messagerie industrielle —

Partie 3: Norme d'accompagnement pour la robotique

Reference number
ISO/IEC 9506-3:1991(E)

Contents

Figures

Tables

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 9506-3 was prepared by Joint Technical Committee ISO/TC 184, *Industrial automation systems and integration*.

ISO/IEC 9506 consists of the following parts, under the general title *Industrial automation systems — Manufacturing message specification*:

— *Part 1: Service definition*

— *Part 2: Protocol specification*

— *Part 3: Companion standard for robotics*

Annexes A and B are for information only.

Introduction

This part of ISO/IEC 9506 is intended to be used in an open communication system employing robots and robotic systems connected to a communication network conforming to the OSI model (ISO 7498). This part of ISO/IEC 9506 also recognizes that the robot can act as a controller (client) to devices connected to it such as vision systems and grippers. Client conformance for communication to such devices is not defined by this part of ISO/IEC 9506. Conformance requirements for communication to such devices are defined by the companion Standard appropriate to that device or by ISO/IEC 9506-1 and ISO/IEC 9506-2.

This part of ISO/IEC 9506 does define conformance requirements for the robot when used in a network with multiple clients. The messages are described using the method defined in ISO 8824.

This part of ISO/IEC 9506 provides a description of several conformance classes including a base class. This base class is considered as the minimum conformance for robots connected as a "slave" or server to a host computer or client device on the network. The base class forms the "kernel" of conformance for robots in these types of networks. All other conformance classes will be additions to the base class. This part of ISO/IEC 9506 also provides the robot specific services and protocol including the abstract syntax notation for protocol elements which are undefined in the MMS-General-Module.

This part of ISO/IEC 9506 also recognizes that the robot can act as a controller to devices connected to it such as vision systems and grippers. This part of ISO/IEC 9506 identifies the requirements for communications in such a manner but does not identify MMS service and protocol conformance requirements for the robot when acting in a client role. These requirements are identified by the companion standard covering the device to which the robot intends to communicate.

MMS is intended to be used with yet other standards designed to achieve a systematic and uniform approach to Open Systems Interconnection of Information Processing Systems as defined in ISO 7498. As such, MMS is positioned within the application layer of the OSI model. It defines the Application Service Element and the protocol required to extend information systems networks to the programmable control devices of the automated factory environment. The services defined by MMS are generic and intended to be referenced by the companion standards, each of which is oriented towards a more specific class of application.

This part of ISO/IEC 9506 recognizes that safe operation of robots is required at all times. Safety requirements for robots are specified in ISO DIS 10218. All robot actions delineated in this part of ISO/IEC 9506 are permissible under the safety standard.

Implementation of this part of ISO/IEC 9506 requires a minimum implementation of MMS. This is covered in Clause 9 which references the conformance requirements of ISO/IEC 9506-1 and 2. Implementers of MMS for robots and robotic systems should have a thorough understanding of MMS for proper implementation of this part of ISO/IEC 9506. Implementers should also have a thorough understanding of the modeling, services and protocol defined in this part of ISO/IEC 9506. Users of robots and robotic systems are directed to the clauses on modeling and services found in this document.

For the purpose of this part of ISO/IEC 9506, the term "robot" means "manipulating industrial robot" as defined in ISO/TR 8373. As used in this part of ISO/IEC 9506, a robot will generally refer to the manipulator together with its control system and any ancillary equipment, devices, sensors, or communications links, necessary for the robot to perform its task. Figure 1 illustrates the elements of the robot system as described in this part of ISO/IEC 9506. Since the definitions of ISO/TR 8373 only describe robot systems with a single arm and this part of ISO/IEC 9506 anticipates robots with multiple arms operating in a coordinated fashion, these definitions have been generalized. The term "robot system controller" will refer to the (single) task program operating with the (possible multiple) control program of the robot arm(s) of the system.

"MMS services" refers to the abstract services defined in ISO/IEC 9506-1 and "MMS protocol" refers to the protocol defined in ISO/IEC 9506-2.

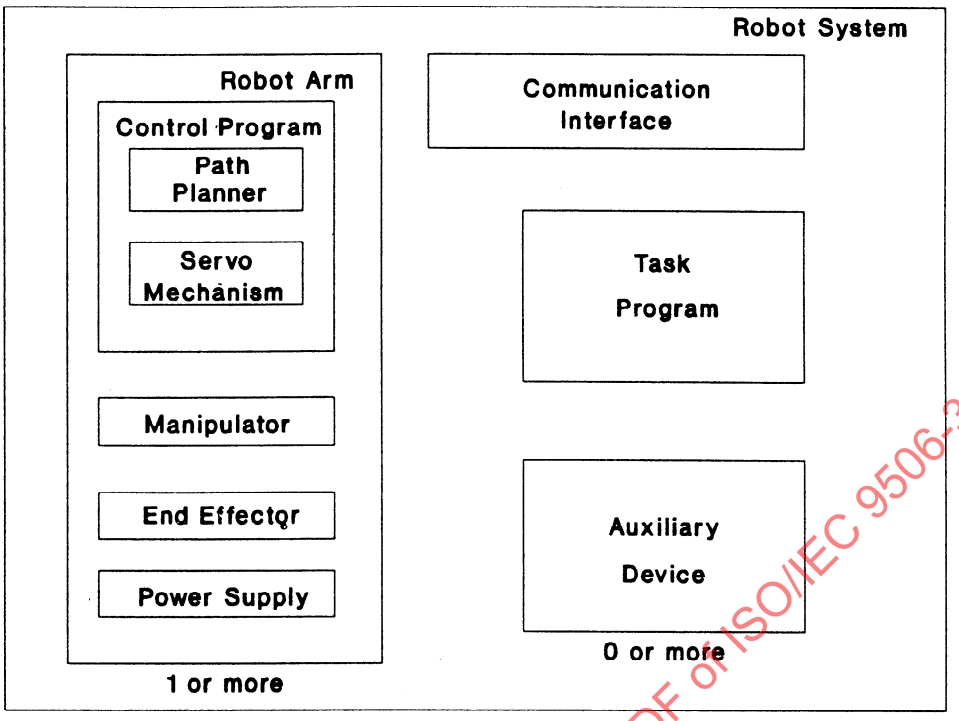

Figure 1 Robot system

This page intentionally left blank

Industrial automation systems — Manufacturing message specification —

Part 3:
Companion standard for robotics

1 Scope

This part of ISO/IEC 9506:

a) describes the model of a robot and how the attributes of the robot are mapped onto the attributes of a Virtual Manufacturing Device (VMD),

b) defines the robot specific services and protocol including the abstract syntax notation for protocol elements requiring companion standard specification by MMS,

c) defines robot specific standardized objects,

d) provides a description of conformance classes including a base class and several enhanced classes.

Definitions are provided of the services and protocol of robots operating as a server in the abstract syntax defined in this part of ISO/IEC 9506. The semantics of MMS services performed by robots while communicating under other abstract syntaxes are not defined by this part of ISO/IEC 9506. This part of ISO/IEC 9506 does not identify MMS service and protocol conformance requirements for a robot acting in a client role. These requirements are intended to be identified by the companion standard covering the device to which the robot intends to communicate.

2 Normative References

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 9506. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 9506 are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of ISO and IEC maintain registers of currently valid international standards.

ISO/IEC 9506-1:1990	*Industrial automation systems - Manufacturing Message Specification - Part 1 - Service definition*
ISO/IEC 9506-2:1990	*Industrial automation systems - Manufacturing Message Specification - Part 2 - Protocol specification*
ISO 7498:1984	*Information processing systems - Open Systems Interconnection - Basic Reference Model*

ISO/TR 8509:1985	*Information processing systems - Open Systems Interconnection - Service Conventions*
ISO 8824:1987	*Information processing systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)*
ISO 8571:1988	*Information processing systems - Open Systems Interconnection - File Transfer, Access and Management*
ISO 8649:1988	*Information processing systems - Open Systems Interconnection - Service definition for the Association Control Service Element*
ISO 8650:1988	*Information processing systems - Open Systems Interconnection - Protocol specification for the Association Control Service Element*
ISO DIS 10218[1]	*Manipulating Industrial Robots - Standard for Safety*
ISO/TR 8373:1988	*Manipulating Industrial Robots -Vocabulary*
ISO 9787:1990	*Manipulating Industrial Robots - Coordinate Systems and Motions*

[1] To be published

3 Definitions

3.1 General definitions

Clause 3 of ISO/IEC 9506-1:1990 and clause 3 of ISO/IEC 9506-2:1990 list a number of terms defined in ISO 7498, in ISO/TR 8509, and in ISO 8824, as well as its own definitions. ISO/TR 8373 and ISO 9787 also define a number of terms used in this part of ISO/IEC 9506. These definitions are included in this part of ISO/IEC 9506 by reference.

3.2 Specific definitions

3.2.1 control program:

The inherent set of control instructions which defines the capabilities, actions, and responses of a robot system. This type of program is fixed and usually not modifiable by the user.

3.2.2 cycle:

A single execution of a task program.

3.2.3 local control:

A boolean value which indicates whether or not it is possible for remote operations to effect changes in the state of the MMS server. If local control is true, remote operations cannot change the state of the server.

Note: This definition is appropriate to MMS Companion Standards and is different from that contained in ISO DIS 10218.

3.2.4 manipulating industrial robot [robot]:

An automatically controlled, reprogrammable, multi-purpose, manipulative machine with several degrees of freedom, which can be either fixed in place or mobile for use in industrial automation applications.

Note: For the purposes of the remaining clauses of this part of ISO/IEC 9506, the term "robot" will mean "manipulating industrial robot".

3.2.5 manipulator:

A machine, the mechanism or which usually consists of a series of segments jointed or sliding relative to one another, for the purpose of grasping and/or moving objects (pieces or tools) usually in several degrees of freedom. It may be controlled by an operator, a programmable electronic controller, or any logic system (for example cam device, wired, etc.).

3.2.6 motion enabled:

A boolean value which if TRUE indicates that a valid command presented to the control program of a robot arm will result in motion of the arm.

3.2.7 pose:

Combination of position and orientation of a part of a robot (for example its mechanical interface) or of a workpiece in a coordinate system.

3.2.8 remote operation:

An operation involving data acquisition or control operating over an OSI communication network using MMS services.

3.2.9 robot arm:

As used in this part of ISO/IEC 9506, a manipulator, an end effector, its power supply, and the control program which controls the manipulator.

3.2.10 robot system:

A robot system includes:

- the robot (hardware and software) consisting of the manipulator whether mobile or not; power supply and control system;
- the end-effector(s);
- any equipment, devices, or sensors required for the robot to perform its task;
- any communication interface that is operating and monitoring the robot, equipment, or sensors, as far as these peripheral devices are supervised by the robot control system.

3.2.11 robot system controller:

The entire control system of the robot, consisting of the (single) task program and the control program(s) for the robot arm(s) and the auxiliary device(s).

3.2.12 step:

An atomic element of task program execution. It may or may not involve robot motion.

Note: The concept of a step is dependent on the robotic programming language.

3.2.13 task program:

The set of motion and auxiliary function instructions which define the specific intended task of the robot system; this type of program is normally generated by the user.

Note: an application is a general area of work, a task is specific within the application.

4 Abbreviations

The following abbreviations are used in the text of this part of ISO/IEC 9506.

ACSE:	Association Control Service Element
ASE:	Application Service Element
ASN.1:	Abstract Syntax Notation One
C:	conditional parameter
CBB:	conformance building block
Cnf:	confirm
CS:	companion standard
DIS:	Draft International Standard
FTAM:	File Transfer, Access, and Management
Ind:	indication
I/O:	input/output
IS:	International Standard
M:	mandatory
MICS:	Mechanical Interface Coordinate System
MMS:	Manufacturing Message Specification
OSI:	Open System Interconnection
PDU:	protocol data unit
PICS:	Protocol implementation conformance statement
Req:	request
Rsp:	response
S:	Selection
TR:	Technical Report
U:	user option parameter
VMD:	Virtual Manufacturing Device

5 Robot application description

5.1 Manufacturing configurations

5.1.1 General considerations

The conformance classes which are defined in clause 9 of this part of ISO/IEC 9506 are described independently of the configurations in which they are used. The configurations described in this subclause are tutorial in nature, and are included to give insight into the rationale which underlies this part of ISO/IEC 9506. Actual configurations can exhibit characteristics of several of these configurations simultaneously.

In a remote communication environment, one node is referred to as the client, the other node is referred to as the server. A host connected to a robot is considered to be a client of the robot. The host generally gives commands to, and monitors functions of, the robot. The robot is considered the server to the host. In the case of the robot connected through an OSI communication channel to intelligent peripherals, e.g., such devices as grippers, vision

systems, or other robots, the robot is viewed as a client with respect to such devices. The connections between devices described in this part of this Standard are considered to be logical connections.

Although operator panels and teach pendants can be used to direct operation of a robot, they are not considered clients in the present sense since they are not connected to the robot through OSI communication channels. Rather, they are considered part of the robot server.

This part of ISO/IEC 9506 does not impose any requirements on the client configurations when the client communicates with a robot. It requires only that the client has the capability to send the appropriate requests to the robot and to receive responses from the robot.

5.1.2 Configuration one: Robot server, single client

This configuration consists of one client (e.g., host computer) controlling, or in communication with, one robot (see Figure 2). The client, or host, sends requests to the robot, or server, to which the robot should respond. The robot can include its own subsystems such as vision and gripper controls. These subsystems will not be directly controlled via MMS and are outside the scope of Configuration one.

In simple implementations of Configuration one, there need be only one MMS association. In more sophisticated implementations, there can be multiple concurrent MMS associations.

An example is the case where two MMS associations are used between the host and the robot for increased throughput.

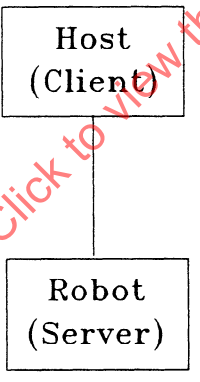

Figure 2 Robot server/single client

5.1.3 Configuration two: Robot server, many clients

In configuration two, the robot is a server, but there are many clients (host computers). See Figure 3. Support for multiple concurrent associations is now required on the part of the robot server, since any of the client hosts can initiate an association.

A multiple client configuration requires a mechanism for taking and relinquishing control of the robot. Without this capability, there is no means for preventing two or more clients from attempting to control the robot simultaneously.

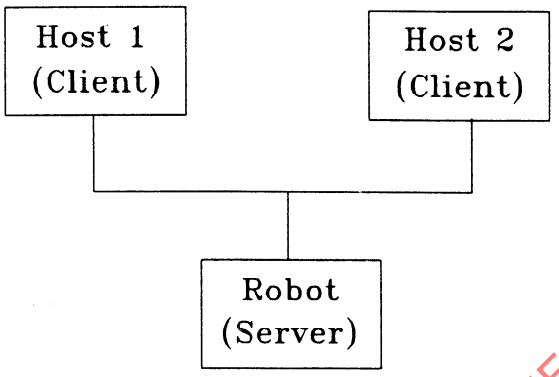

Figure 3 Robot server/many clients

An example of this configuration is a host performing robot control and a second host monitoring the robot as part of a larger system.

5.1.4 Configuration three: Robot client

In configuration three, the robot is a client to one or more devices (see Figure 4). It is also possible that the robot is simultaneously a server to one or more host clients as in configurations one and two respectively. The robot in a client capacity requires the ability to act as an initiator of requests, rather than just as a responder.

This part of ISO/IEC 9506 addresses only the interactions between a system acting as a client and robots acting as servers, and does not define the interactions with other devices such as vision systems, grippers, etc.. These interactions can only be defined based on the requirements of those devices.

An example of this configuration is a robot acting as a client to another system acting as a file server.

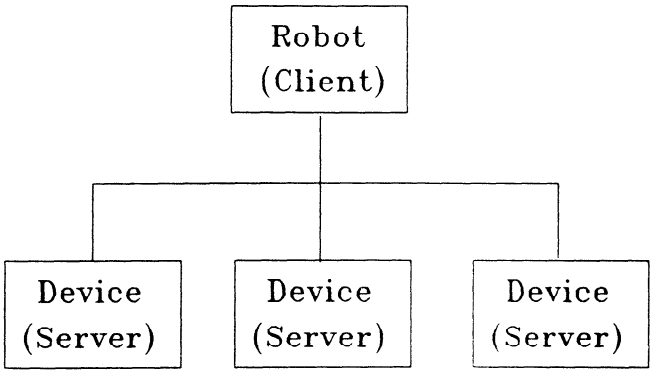

Figure 4 Robot client

5.1.5 Configuration four: Peer to peer

In configuration four, several robots are considered peers, and can act as both clients and servers (see Figure 5). In this configuration each robot includes both client and server capabilities.

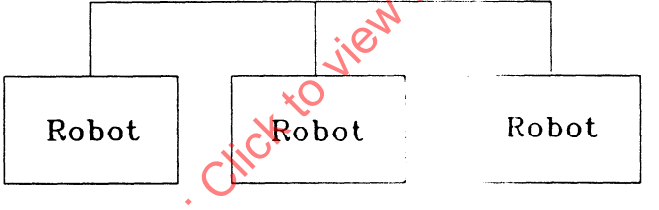

Figure 5 Peer to peer

5.2 Robot specific model

5.2.1 Robot physical model

5.2.1.1 General

This part of ISO/IEC 9056 describes an abstract model of a robot system. The attributes of this model are required to describe the activity of the robot as viewed from a communications channel. Any real robot system can have many more characteristics than are described here. Nor is it required that a robot system have all of the attributes described in this part of ISO/IEC 9506.

A robot system is composed of one or more robot arms together with a robot system controller. There can also be auxiliary devices whose activities are coordinated with the robot arm but are physically and logically separate from the robot arm. In particular, the safety interlocks are one such auxiliary device of which the Emergency Stop Button can be considered to be a component.

5.2.1.2 Robot arm

5.2.1.2.1 Robot arm subsystems

The central element of a robot system is the robot arm which is composed of the manipulator, its power supply, the robot arm control program, and the end effector. The focus of this part of ISO/IEC 9506 is the remote operation of this robot arm, together with coordinated control of the auxiliary devices.

The manipulator is composed of a set of mechanical linkages and joints. Each link along with its associated joint comprises an axis of the robot. The joint is driven by an actuator which is controlled by the robot arm control program. This robot arm control program can be considered to have two main components, a servomechanism and a path planner.

There are characteristics of the robot arm modeled in this part of ISO/IEC 9506 which equally affect both the servomechanism and path planner subsystems. It is essential to know the number of joints of the manipulator and the characteristics of each joint in order to perform the robot task. Knowledge of the nature of the calibration state (calibrated, not calibrated, or calibrating) and power state (arm power on/off) is necessary for proper operation of the robot system.

The flow of information in a robot arm can be described as follows (See Figure 6):

A task program generates the programmed pose of the manipulator.

The programmed pose is then transferred to the path planner subsystem which generates the commanded state expressed in joint values.

The commanded state is then input to the servomechanism subsystem which drives the manipulator.

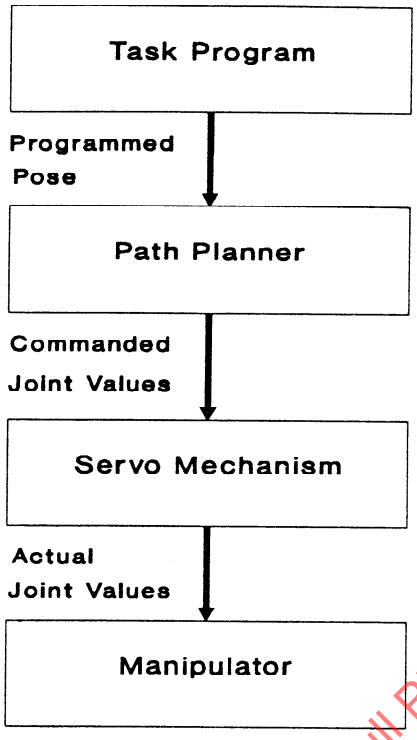

Figure 6 Information flow

5.2.1.2.2 Path planner

The path planner subsystem accepts commands from the task program (or from other local sources) for motion of the manipulator and converts these requests into a time series of servomechanism commands. In effect, the path planner is responsible for converting a desired manipulator trajectory expressed in terms of the position or speed of the tool tip into appropriate servomechanism commands. In order to express these motions in a consistent fashion, a set of coordinate systems are introduced with which to describe the robot arm. These are described in detail in 5.2.2.

The path planner is responsible for monitoring certain characteristics of the manipulator motion and applying desired modifications. Most notable of these is the control of the speed and acceleration of the tool tip.

The programmed speed is the speed normally resulting from the command input to the path planner from the task program of the robot system controller. It is the normal tool tip speed established in the task program.

The speed factor is an override multiplication factor that is applied to alter the speed of the manipulator. The speed factor will alter the programmed speed of the manipulator so that the entire manipulator motion, as directed by the task program, can be executed at a speed which is uniformly adjusted upward or downward from the programmed speed.

The programmed acceleration is the acceleration normally resulting from the commands input to the path planner from the task program of the robot system controller. It is the normal tool tip acceleration established in the task program.

The acceleration factor is an override multiplication factor that is applied to alter the acceleration of the manipulator. The acceleration factor will alter the programmed acceleration of the manipulator so that the entire manipulator motion, as controlled by the task program, can be executed with an acceleration which is uniformly adjusted upward or downward from the programmed acceleration.

Additional attributes of the path planner include a set of input values and output values. The input values to the path planner describe the programmed state of the manipulator expressed in Euclidean space including position, velocity, and acceleration. The output values from a path planner describe the commanded state of the manipulator expressed in joint space.

5.2.1.2.3 Robot arm servomechanism

The robot arm servomechanism subsystem consists of a coupled set of servomechanisms, one for each joint of the manipulator. The joints of a manipulator can be of revolute or prismatic type. Each joint can be individually calibrated, can have upper and lower bounds on travel, can have a brake mechanism, and has an associated set of link parameters relating it to the other joints in the kinematic chain which comprise the robot arm. Each joint is controlled by a servomechanism which sends control signals to the joint actuator and may obtain feedback information from joint sensors. The servomechanism issues commands to drive the joint to a desired value and monitors the actual value as it moves.

5.2.1.3 Auxiliary devices

The robot auxiliary devices are robot task related subsystems that are directly controlled by the robot system controller. The control of auxiliary devices is integrated into the control of the robot task. The information used in auxiliary device control, for example specific parameters of the auxiliary devices, are an integral part of the overall parameters of the robot task. Control of auxiliary devices can be implemented in one or more robot programs.

Note: Examples of auxiliary devices are safety interlocks, weld controllers, paint systems, water jet cutter systems, vision systems, sensors, and grippers.

5.2.2 Robot coordinate system

5.2.2.1 Conventions

The robot coordinate systems used by this part of ISO/IEC 9506 are the world, base and mechanical interface coordinate systems. These standardized coordinate systems are described in ISO 9787. Additional robot coordinate systems are described in the 5.2.2.2 and 5.2.2.3 as an aid to the understanding this part of ISO/IEC 9506.

The World Coordinate System establishes a fixed frame of reference for the entire manufacturing operation and is commonly used for work cell layout. The definition of this coordinate system will generally be made by the user institution. The Base Coordinate System establishes the location of the mounting plate of the robot and is generally supplied by the manufacturer of the robot. The Mechanical Interface Coordinate system is a moving coordinate system attached to the last link of the manipulator which establishes the location of the end-effector relative to the robot base position.

Two commonly used coordinate systems are introduced in addition to the standardized coordinate systems; the Tool Coordinate System, and the User Coordinate System. The Tool Coordinate System is used to define the position of a robotic tool (end effector). The User Coordinate System is used to provide an alternative frame of reference for the robot arm other that the Base Coordinate System. Figure 7 illustrates the robot coordinate system conventions.

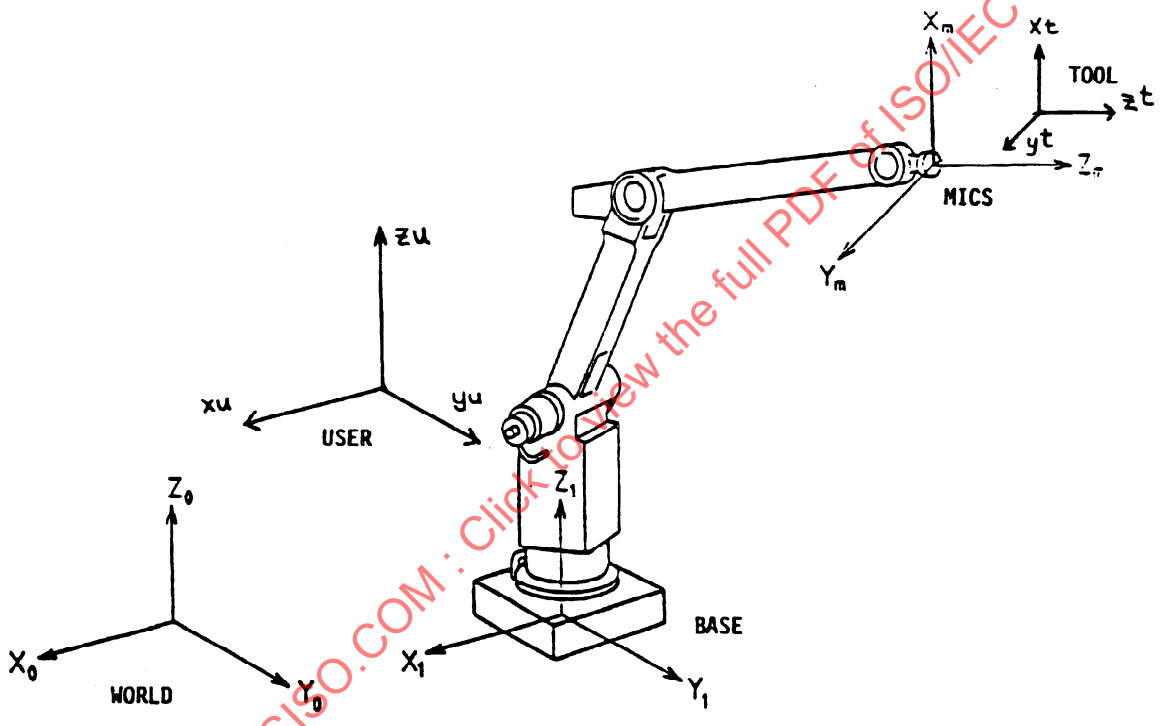

Figure 7 Robot coordinate systems

5.2.2.2 Tool Coordinate System (X_t, Y_t, Z_t)

The origin of the Tool Coordinate System is positioned at a implementor defined point on the robotic tool. The + Z_t and + X_t axes are implementor defined and perpendicular to each other. The + Y_t axis is determined by the right hand rule.

An object within the Tool Coordinate System is represented by a transformation of the form (x_t, y_t, z_t, a_t, b_t, c_t) where:

x_t represents a translation along the X_t coordinate axis.

y_t represents a translation along the Y_t coordinate axis.

z_t represents a translation along the Z_t coordinate axis.

a_t represents a counter clockwise rotation about the positive X_t coordinate axis.

b_t represents a counter clockwise rotation about the positive Y_t coordinate axis.

c_t represents a counter clockwise rotation about the positive Z_t coordinate axis.

The transformation follows a right-handed rule sign direction convention.

5.2.2.3 The User Coordinate System (X_u, Y_u, Z_u)

The origin of the User Coordinate System is positioned at a implementor defined point. The + Z_u and + X_u axes are implementor defined and perpendicular to each other. The + Y_u axis is determined by the right hand rule.

An object within the User Coordinate System is represented by a transformation of the form (x_u, y_u, z_u, a_u, b_u, c_u) where:

x_u represents a translation along the X_u coordinate axis.

y_u represents a translation along the Y_u coordinate axis.

z_u represents a translation along the Z_u coordinate axis.

a_u represents a counter clockwise rotation about the positive X_u coordinate axis.

b_u represents a counter clockwise rotation about the positive Y_u coordinate axis.

c_u represents a counter clockwise rotation about the positive Z_u coordinate axis.

The transformation follows a right-handed rule sign direction convention.

5.2.2.4 Transformations of coordinate systems

In the course of describing robot motion, it is necessary to describe the arbitrary position and orientation of objects in space. This description can be made with respect to any of the coordinate systems described above. Mathematically, the description of the position and orientation of an object with respect to some coordinate system is equivalent to the description of a transformation from the referenced coordinate system to a coordinate system embedded in the object. Thus, transformations assume a large role in the mathematics of robot motion.

Of the five coordinate systems described above, four are often used as reference coordinate systems in which to describe other objects. The Tool Coordinate System is the coordinate system embedded in the object whose position and orientation is needed. This part of this Standard identifies five transformations which are useful to describe the kinematic relationship of the components of a robot system. These are:

- o The Base Coordinate System with respect to World Coordinate System transformation.
- o The Mechanical Interface Coordinate System with respect to Base Coordinate System transformation.
- o The User Coordinate System with respect to Base Coordinate System transformation.
- o The Tool Coordinate System with respect to the User Coordinate System transformation.
- o The Tool Coordinate System with respect to the Mechanical Interface Coordinate System transformation.

5.3 Robot specific functions

5.3.1 Control systems

5.3.1.1 Robot system controller

An essential element of the robot system is the robot system controller which represents the intelligence governing the operation of the robot. The robot system controller is usually implemented through the adaptation of a multi-purpose computer running one or more computer programs, but it can also be implemented by other means. In computer based implementations, it may be the case that the underlying computer is used for more functions than the control of the robot but this part of ISO/IEC 9506 is solely concerned with those aspects of the computer operation which relate to robot operation. Robot operation, in this sense, includes the operation of all auxiliary devices which are necessary for the robot to perform its function.

An example of such a computer system which lies beyond the scope of this part of ISO/IEC 9506 is an integrated cell controller which directs a robot's activity as one part of its operation.

5.3.1.2 Coordination of robot control

In many cases control over the robot operation can be exercised from multiple locations. The introduction of communication facilities adds another potential point of control. In order to preclude conflicts in control, the robot is assumed to possess a method of assigning control uniquely to one control point. This control assignment functions as a mutually exclusive semaphore.

5.3.1.3 Robot operation states

5.3.1.3.1 Robot operation state diagram

The robot system is modeled by a finite state automaton as described in Figure 8.

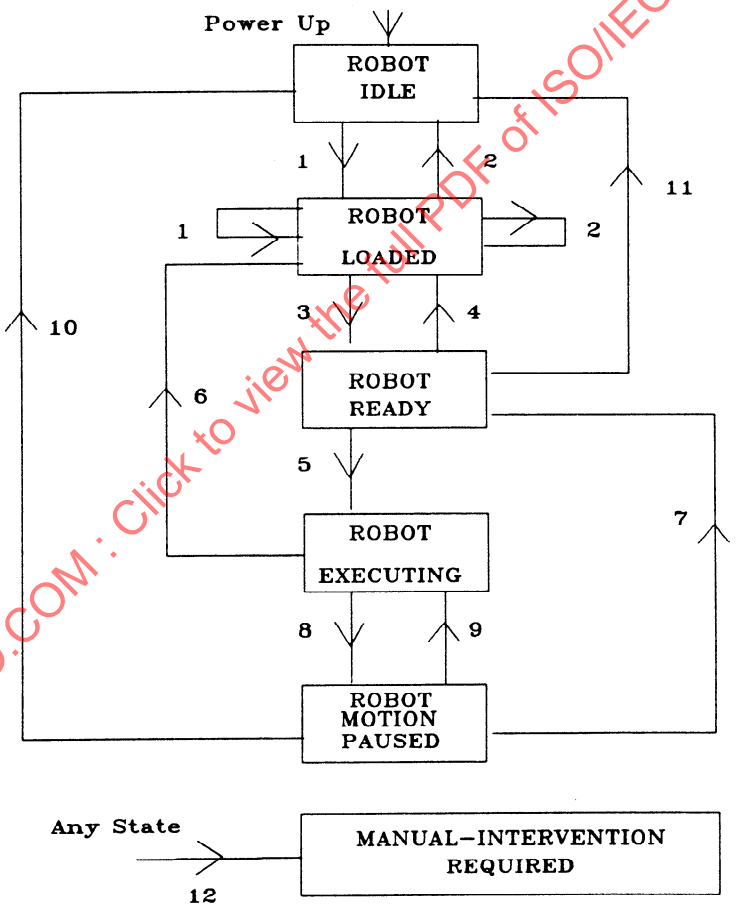

Figure 8 Robot operation states

Transition Descriptions for Figure 8.

1.	Load	Some task program is placed into the robot system controller.
2.	Clear	The task program is removed from the robot system controller.
3.	Assign	One loaded task program is made ready for execution.
4.	Deassign	The inverse of Assign.
5.	Program Start	The robot is placed into the ROBOT-EXECUTING state.
6.	End of Execution	The robot motion is halted and the robot is returned to the ROBOT-LOADED state.
7.	Program Reset	A paused robot is returned to the ROBOT-READY state.
8.	Pause	The robot has temporarily ceased motion.
9.	Continue	A paused robot has been returned to the ROBOT-EXECUTING state.
10.	Clear(abort)	Same as clear, except the task program has not completed.
11.	Clear(abandon)	Same as clear, except the task program has not started.
12.	Emergency Action	Places the robot into a state in which manual intervention is required.

5.3.1.3.2 Robot operation state description

5.3.1.3.2.1 ROBOT-IDLE

The ROBOT-IDLE state corresponds to the state of the robot when the robot has no task program available within the system. This state will occur when the robot is initially powered up or when it has completed a task program execution and the task program is removed from the system.

5.3.1.3.2.2 ROBOT-LOADED

The ROBOT-LOADED state corresponds to the state of the robot when one or more task programs are available within the system.

5.3.1.3.2.3 ROBOT-READY

The ROBOT-READY state indicates that one of the task programs available for execution by the robot has been assigned.

5.3.1.3.2.4 ROBOT-EXECUTING

In the ROBOT-EXECUTING state the robot is in operation and its motion is enabled.

5.3.1.3.2.5 ROBOT-MOTION-PAUSED

In the ROBOT-MOTION-PAUSED state motion is not enabled. The task program of the robot can be running or it can be stopped.

5.3.1.3.2.6 MANUAL-INTERVENTION-REQUIRED

In this state, all motion has ceased. In order for any other action to be performed, some local action is required by a machine operator. Following this local action, the robot may be in any of the other states.

5.3.1.3.3 Robot operation state transitions

5.3.1.3.3.1 General

In the description of the transitions which follow, it is assumed that all these state transitions may occur through local action. Many of these transitions can also be accomplished through remote actions. The local actions which cause these transitions will not be further described. Remote actions using MMS services which cause these transitions are described in clause 6.

5.3.1.3.3.2 Load

The Load transition describes the act of loading a task program into the robot system controller. The robot system controller may have several task programs within its memory; for the purposes of this transition, however, only task programs which can operate and control the motion of the robot arm are considered. If there are no such programs within the robot system controller, this transition will take the robot from ROBOT-IDLE to ROBOT-LOADED state. If there is at least one such task program in memory, the robot remains in ROBOT-LOADED state as a result of this transition.

5.3.1.3.3.3 Clear

This transition occurs when a task program is removed from the robot system controller. If this task program is the last or the only such program, the robot is placed in ROBOT-IDLE state. Otherwise, it continues in ROBOT-LOADED with the remaining task programs.

5.3.1.3.3.4 Assign

The process of assigning one of the possible task programs for actual execution will place the robot in ROBOT-READY state. For robots which can only contain one task program the Load and Assign transitions may be combined into a single operation.

5.3.1.3.3.5 Deassign

To move the robot from the ROBOT-READY state to the ROBOT-LOADED state requires a Deassign transition. This is the inverse of the Assign transition. Some robots may appear to allow an Assign transition from the ROBOT-READY state. This can always be modeled as a Deassign followed by an Assign transition.

5.3.1.3.3.6 Program Start

The Program Start is the fundamental operation of placing the robot in motion. As a result of this transition, the robot moves into the ROBOT-EXECUTING state.

5.3.1.3.3.7 End of Execution

As a result of local actions within the robot system controller, the robot may move out of the ROBOT-EXECUTING state into the ROBOT-LOADED state. Before the robot can be placed in the ROBOT-EXECUTING state again, an Assign transition is required.

5.3.1.3.3.8 Program Reset

A robot which is in the ROBOT-MOTION-PAUSED state can be moved to the ROBOT-READY state by the Program Reset transition.

5.3.1.3.3.9 Pause

The Pause transition causes the robot to cease its movement. The task program may continue to execute or it may stop.

5.3.1.3.3.10 Continue

The Continue transition will move a robot from the ROBOT-MOTION-PAUSED state back into the ROBOT-EXECUTING state.

5.3.1.3.3.11 Clear (Abort)

The Clear transition can also be made from the ROBOT-MOTION-PAUSED state. In this case, the current execution of the task program is aborted (i.e. abnormally terminated) and the assigned task program is deassigned. All potential task programs are removed from the robot system controller.

5.3.1.3.3.12 Clear (Abandon)

The Clear transition can also be made from the ROBOT-READY state. In this case, the current execution of the task program is aborted (i.e. abnormally terminated) and the assigned task program is deassigned. All potential task programs are removed from the robot system controller.

5.3.1.3.3.13 Emergency Action

The Emergency Action transition may occur from any state into the MANUAL-INTERVENTION-REQUIRED state as a result of local or remote action. Recovery from this state can never be automatic and shall require the explicit action of a human operator to place the robot into some other state.

5.3.1.4 Local control

In addition to the Robot State, the robot can be described as either being in local control or not being in local control. Being in local control means that some local agent, e.g. a human operator at a control console or at a teach pendant or equivalent, is exercising control over what motions the robot may make or over what task programs the robot may run. If the robot is not in local control, the robot can still be running a task program, but the ability to stop or otherwise modify that running task program does not reside in any local agent. Robots which are capable of having remote controllers may then have such remote controllers take control of the robot.

5.3.2 Task program execution - cycles and steps

The operation of a robot may be decomposed into cycles and steps. A cycle is a sequence of operations which is repeated regularly. Each cycle of robot operation completes the task for which the robot is programmed. The cycle is the natural unit of robot operation.

Within a cycle, more atomic actions of the robot can be distinguished. These atomic actions are referred to as steps, and correspond to some elementary motion of the robot or to some elementary action of the task program or of one of the auxiliary devices. Fundamental to this notion of steps is the idea of sequential operation; a robot is composed of systems elements each of which performs elementary operations in a time ordered sequence. Not all robot systems allow control based on individual steps; for those that do, this part of ISO/IEC 9506 provides control procedures for handling individual steps.

When the robot begins operation, it can be set to operate for a fixed number of cycles, or (if in step mode) for a fixed number of steps within a cycle. This latter mode is normally used only for debugging. Alternatively, the robot can be placed in operation to perform the same cycle repetitively, without count, until some external event terminates the process.

5.3.3 Calibrate

In addition to normal operation, most robots have an auxiliary function available for calibration of the axes of the robot. This function is considered to operate in a manner similar to a normal task program, subject to the same state diagram (Figure 8) except that the presence of the calibration function does not place the robot in the LOADED state.

6 Robot application specific context mapping

6.1 Mapping the robot model to the VMD object

6.1.1 Description of mapping procedure

This clause relates the general model of the robot, developed in clause 5, to the abstract model of the Virtual Manufacturing Device (VMD) described in ISO/IEC 9506-1. The uniform application of these abstract concepts to real systems is essential if the final goal of inter-operability is to be achieved. While this part of ISO/IEC 9506 cannot anticipate every variant of robot systems which are or could be manufactured, it provides general guidelines for associating elements of real systems with the abstract model that should be applicable for most robots.

The mapping process is a multi-step procedure including the following steps:

a) An abstract model of a set of real physical systems is first constructed. In this case, this is the model developed in clause 5.

b) From the abstract model a set of abstract objects and attributes is constructed.

c) The set of attributes is related to a set of attributes of an MMS abstract object. If necessary, extensions to the MMS abstract object are defined to permit this relation.

d) Finally, the device specific abstract object is related to the MMS abstract object.

This process is illustrated in Figure 9.

In order to accommodate all the features of the robot model, extensions to the model of Program Invocations are developed in 6.3. These extensions permit the description of interworking between two classes of Program Invocations referred to as 'Controlling' and 'Controlled'. The Controlled Program Invocation is used to model the operation of the physical device controllers including the robot arm control program. The Controlling Program Invocation is used to model that specific Program Invocation which directs the operation of the robot system. While a VMD can have several Controlling Program Invocations in existence, only one is identified as being selected for operation at any specific time. In addition to the association of physical devices with Controlled Program Invocations, each physical device is also mapped to a predefined Domain. This mapping is described in 6.2.

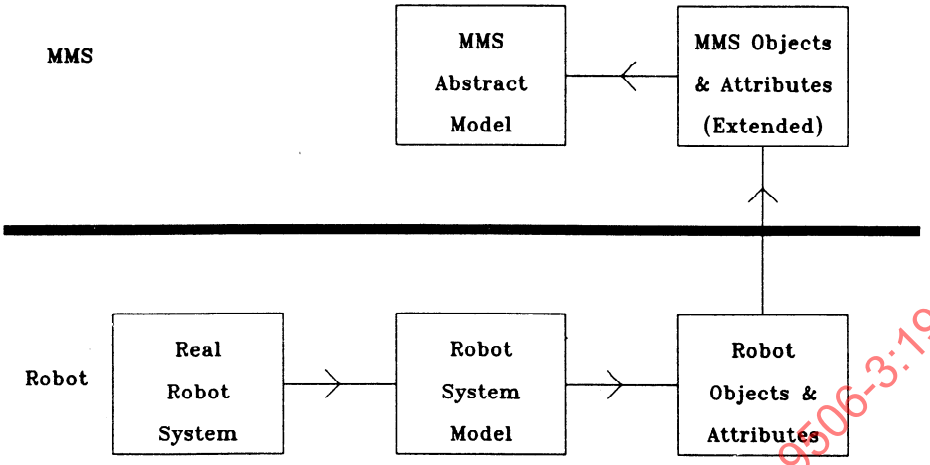

Figure 9 MMS mapping process

6.1.2 Robot system characterization

6.1.2.1 Single arm robots

From the modeling point of view, the simplest robot systems are those with only one arm. The VMD is associated with the entire robot system, robot arm, robot system controller, and any auxiliary devices. The robot arm control program is modeled as a Controlled Program Invocation associated with the motion hardware.

6.1.2.2 Multiple coordinated arm robots

There are robot systems which employ more than one arm. If a system has a single controlling intelligence such that the motion of the several arms is always conducted in a coordinated fashion, then such a system may be modeled as a single VMD. In this case, there will be a single Controlling Program Invocation which controls the motions of both arms.

This modeling is independent of the visual appearance of the robot system. Two separate pieces of hardware can be mounted at two separate locations. If the action of the two is always in close coordination, a single VMD is the proper modeling concept.

6.1.2.3 Multiple independent arm robots

There are systems in which multiple arms are employed within the same location, but for which the motion of the arms is independent, that is, there is a separate controlling intelligence for each arm. In such cases, the system should be modeled as having multiple VMDs; one for each controlling intelligence. In this case, there will be a

single Controlling Program Invocation which controls motion in each VMD. Note that the distinction between the two cases depends greatly on the configuration of the robot system controller. It is possible that the same piece of hardware can be configured either as a multiple independent arm robot or as a multiple coordinated arm robot. In such cases, the appropriate model will be different for each configuration.

6.1.2.4 Auxiliary devices

The tasks which operate the auxiliary devices do so in a coordinated manner with the robot arm control program and can be thought of as part of a single controlling task. They are modeled as additional Controlled Program Invocations.

6.1.3 The Robot VMD

6.1.3.1 Robot VMD attribute model

Given the association of the VMD with the robot system, additional attributes of the VMD (beyond those given by ISO/IEC 9506-1) have been defined to describe the robot. Since the essential nature of the VMD is the coordinated control of the robot arm and the auxiliary devices, the Controlling Program Invocation which provides this control is identified in the VMD. The Controlling Program Invocation will, during the course of its execution, produce outputs which will be used as input to the Controlled Program Invocations.

The Robot Operation State is derived from the condition of the hardware (whether motion is enabled or not) as represented by a set of Controlled Program Invocations, and from the state of the selected Controlling Program Invocation (See 6.3).

Object: VMD

All MMS defined Attributes
Attribute: Safety Interlocks Violated (TRUE, FALSE)
Attribute: Robot Operation State (ROBOT-IDLE, ROBOT-LOADED, ROBOT-READY, ROBOT-EXECUTING, ROBOT-PAUSED, MANUAL-INTERVENTION-REQUIRED)
Attribute: Any Physical Resource Power On (TRUE, FALSE)
Attribute: All Physical Resources Calibrated (TRUE, FALSE)
Attribute: Local Control (TRUE, FALSE)
Attribute: Reference to Selected Controlling Program Invocation

6.1.3.2 Robot VMD attribute description

6.1.3.2.1 Safety Interlocks Violated

This attribute, of type boolean, indicates the state of the safety interlocks. If the value of this attribute is TRUE, the safety interlocks have been violated since the robot system was last reset. The method of resetting the safety interlocks is a local matter.

6.1.3.2.2 Robot Operation State

This attribute is the state of the robot system as described in 5.3.1.3. The relationship of this attribute to the physical devices and to the state of the Controlling Program Invocations is defined in 6.3.4.

6.1.3.2.3 Any Physical Resource Power On

This attribute, of type boolean, indicates whether (TRUE) or not (FALSE) any physical resource has power applied to it. This attribute is the logical 'OR' of a similar primitive attribute of each physical resource in the system.

6.1.3.2.4 All Physical Resources Calibrated

This attribute, of type boolean, indicates whether (TRUE) or not (FALSE) all the physical resources in the system which have calibration attributes have the value equal to CALIBRATED (See 6.2.2.4 and 6.2.3.4).

6.1.3.2.5 Local Control

This attribute, of type boolean, indicates whether (TRUE) or not (FALSE) some local agent has control of any of the physical resources of the system. Having control indicates the ability to cause actions which change the physical resources and hence the attributes which represent those resources. Local Control can reflect either a human operator or some automatic procedure, either of which inhibit the assertion of control remotely.

If Local Control is FALSE, control may reside in some remote agent. This condition can also reflect a condition in which the robot is running under no direct control other than its task program.

The value of the Local Control attribute, the Robot VMD State, and the VMD Logical Status are interrelated. Table 1 illustrates the relationship between these attributes.

Table 1 Local control

Local Control	Robot Operation State	VMD Logical Status
TRUE	Any	NO-STATE-CHANGES-ALLOWED or LIMITED-SERVICES-PERMITTED or SUPPORT-SERVICES-ALLOWED
FALSE	MANUAL-INTERVENTION-REQUIRED	NO-STATE-CHANGES-ALLOWED or LIMITED-SERVICES-PERMITTED or SUPPORT-SERVICES-ALLOWED
	Any other	Any

Note: While in Local Control, the VMD State can continue to change. However, it is a local matter how the state changes occur.

6.1.3.2.6 Selected Controlling Program Invocation

This attribute identifies the Program Invocation which has been selected to control the operation of the robot. This Program Invocation represents the assigned task program (See 5.3.1.3.3.4). This Program Invocation has its Control attribute equal to CONTROLLING (See 6.3.2.3) and has been selected through the use of the Select operation (See 7.4.4). If there is no such Program Invocation identified, this attribute shall have the value NONE.

6.2 Robot specific objects that map to Domains

6.2.1 General

The robot, as modeled in this part of ISO/IEC 9506, consists of a set of physical resources representing the robot arm (or arms) and auxiliary devices. Each physical resource or group of resources shall be mapped to a Domain which represents that resource. Each Domain shall have attributes associated with the control of the resource, state of the resource's power, and may have an attribute which indicates the state of calibration of the resource.

The Robot Arm standardized Domain shall be associated with the robot arm resource, or in the case of multiple arms, one Domain shall be associated with each such arm. This Domain shall include within its definition all peripheral elements which are normally part of the robot arm.

Auxiliary devices which are controlled by separate programs shall be modeled by separate predefined Domains. In particular, the safety interlocks shall be modeled by one such Domain.

These Domains shall be bound to the underlying resources for all data reporting services (that is, for read services). For services which effect control or change of state of the device, the binding shall be moderated by the Local Control attribute of the VMD.

There are no extensions defined for the MMS Domain; the attributes of the robot specific objects shall be realized by Domain specific Named Variable objects defined within the Domain. Naming standards for such Named Variable objects are prescribed in clause 8.

6.2.2 The Robot arm resource

6.2.2.1 Robot arm attribute model

A robot arm includes all elements of the arm that are normally controlled by the robot arm control program. The attributes of the Robot Arm object describe the robot arm from a kinematic and control perspective.

This abstract model contains only those attributes required for remote operation of the robot. Any real robot has many more attributes that are not included in this part of ISO/IEC 9506.

Object: Robot Arm

Attribute : Local Control (TRUE, FALSE)
Attribute : Device Power On (TRUE, FALSE)
Attribute : Device Calibrated (CALIBRATED, NOT-CALIBRATED, CALIBRATING)
Attribute : Motion Enabled (TRUE, FALSE)
Attribute : Number Of Joints - Integer
Attribute : BASE-WORLD - Pose
Attribute : Servomechanism
  Attribute : MICS-BASE - pose
  Attribute : Ordered List Of Joint Description
    Attribute : Joint Type (REVOLUTE, PRISMATIC)
    Attribute : Calibrated (CALIBRATED, NOTCALIBRATED, CALIBRATING)
    Attribute : Joint Brakes (TRUE, FALSE)
    Constraint : Joint Brakes = TRUE
      Attribute : Brakes On (TRUE, FALSE)
    Attribute : Upper Bound - Floating_point
    Attribute : Lower Bound - Floating_point
    Attribute : Joint Servo
      Attribute : Actual Joint Value - Floating_point
Attribute : Path Planner
  Attribute : USER-BASE - Pose
  Attribute : Desired TOOL-USER - Pose
  Attribute : Speed Factor - Floating_point
  Attribute : Programmed Speed - Floating_point
  Attribute : Acceleration Factor - Floating_point
  Attribute : Programmed Acceleration - Floating_point
Attribute : End Effector
  Attribute : ID Number
  Attribute : Tool Descriptor
  Attribute : TOOL-MICS - Pose

Note: Robot terminology normally considers an axis to represent a joint and an associated link. Link information is a local matter. For the purposes of this abstract model joints and axes are identical.

6.2.2.2 Local Control

This attribute, of type boolean, indicates whether (TRUE) or not (FALSE) the robot arm is in local control.

6.2.2.3 Device Power On

This attribute, of type boolean, indicates whether (TRUE) or not (FALSE) the robot arm has power applied to it.

6.2.2.4 Device Calibrated

This attribute indicates whether the robot arm as a unit is calibrated. This attribute has the value CALIBRATED if all joints are calibrated, CALIBRATING if any joint is calibrating, otherwise NOT-CALIBRATED.

6.2.2.5 Motion Enabled

This attribute, of type boolean, indicates whether (TRUE) or not (FALSE) the robot arm (or any part of it) will move if a valid command signal is presented to the path planner. The robot arm may be moving autonomously if and only if the Motion Enabled attribute is TRUE.

6.2.2.6 Number Of Joints

This attribute, of type integer, indicates the number of joints of the robot arm.

6.2.2.7 BASE-WORLD

This attribute, of type Pose, indicates the value of the transformation of the Base Coordinate System with respect to the World Coordinate System.

6.2.2.8 Servomechanism

6.2.2.8.1 MICS-BASE

This attribute, of type Pose, indicates the actual value of the transformation of the Mechanical Interface Coordinate System with respect to the Base Coordinate System. It is derived from the Actual Joint Value attribute and implementation specific parameters.

6.2.2.8.2 Ordered List Of Joint Description

6.2.2.8.2.1 Joint Type

This attribute indicates whether the joint mechanism of the joint is revolute or prismatic.

6.2.2.8.2.2 Calibrated

This attribute indicates whether the joint is CALIBRATED, NOT-CALIBRATED, or CALIBRATING.

6.2.2.8.2.3 Joint Brakes

This attribute, of type boolean, indicates whether (TRUE) or not (FALSE) brakes exist for this joint. If brakes exist (Joint Brakes is TRUE), then the attribute Brakes On exists. The Brakes On attribute, of type boolean, indicates whether (TRUE) or not (FALSE) the brakes are applied to restrict motion.

6.2.2.8.2.4 Upper Bound

This attribute, of type floating point, indicates the upper limit of travel for the joint.

6.2.2.8.2.5 Lower Bound

This attribute, of type floating point, indicates the lower limit of travel for the joint.

6.2.2.8.2.6 Joint Servo - Actual Joint Value

This attribute, of type floating point, indicates the actual value of the joint parameter. The value will be the linear position or angular position depending upon the Joint Type.

6.2.2.9 Path Planner

6.2.2.9.1 USER-BASE

This attribute, of type Pose, indicates the value of the transformation of the User Coordinate System with respect to the Base Coordinate System.

6.2.2.9.2 Desired TOOL-USER

This attribute, of type Pose, indicates the value of the desired pose of the Tool Coordinate System with respect to the User Coordinate System.

6.2.2.9.3 Speed Factor

This attribute, of type floating point, identifies the override speed factor to be applied to the Programmed Speed input to the path planner.

6.2.2.9.4 Programmed Speed

This attribute, of type floating point, identifies the magnitude of the programmed speed of the MICS coordinate system with respect to the Base coordinate system as designated in the robot program.

6.2.2.9.5 Acceleration Factor

This attribute, of type floating point, identifies the override acceleration factor to be applied to the Programmed Acceleration input to the path planner.

6.2.2.9.6 Programmed Acceleration

This attribute, of type floating point, identifies the magnitude of the programmed acceleration of the MICS coordinate system with respect to the Base coordinate system as designated in the robot program.

6.2.2.10 End Effector

6.2.2.10.1 ID Number

This attribute, of type object identifier, identifies uniquely the end effector from all other end effectors within a certain set.

6.2.2.10.2 Tool Descriptor

This attribute provides a text description of the end effector.

6.2.2.10.3 TOOL-MICS

This attribute, of type Pose, indicates the value of the transformation of the Tool Coordinate System with respect to the Mechanical Interface Coordinate System.

6.2.3 Robot auxiliary device resource

6.2.3.1 Auxiliary attribute model

All physical resources separate from the robot arm resource are modeled as auxiliary device resources. The Auxiliary Device object describes the attributes of the physical resource.

Object: Auxiliary Device

Attribute : Local Control (TRUE, FALSE)
Attribute : Device Power On (TRUE, FALSE)
Attribute : Device Calibrated (CALIBRATED, NOT-CALIBRATED, CALIBRATING)

6.2.3.2 Local Control

This attribute,of type boolean, indicates whether (TRUE) or not (FALSE) the auxiliary device resource is in local control.

6.2.3.3 Device Power On

This attribute, of type boolean, indicates whether (TRUE) or not (FALSE) the auxiliary device resource has power applied to it.

6.2.3.4 Device Calibrated

This attribute indicates whether the auxiliary device resource as a unit is calibrated. If calibration does not apply to a particular auxiliary device, then this attribute does not exist.

6.3 Robot specific objects that map to Program Invocations

6.3.1 Robot specific Program Invocations

The model of a Program Invocation is augmented to allow the concept of interworking of two related Program Invocations in the same VMD. In such a pair of interworking Program Invocations, one Program Invocation is referred to as the Controlling Program Invocation, the other as the Controlled Program Invocation. The two Program Invocations are modeled as having an inter-process communication channel available to them over which they pass messages which provide both data and synchronization of the processes. However, such messages are not visible to the remote MMS user except insofar as they cause the state of the Program Invocation to be altered.

The relationship between these two types of Program Invocations is a many to one relationship. A Controlling Program Invocation can control many Controlled Program Invocations; a Controlled Program Invocation can be controlled by only one Controlling Program Invocation at any specific time.

6.3.2 Extensions to the Program Invocation object model

6.3.2.1 Program Invocation attribute model

The model of the Program Invocation given by ISO/IEC 9506-1 shall be augmented by the following attributes:

Object: Program Invocation

Attribute: All MMS Attributes
Attribute: Error Code - Integer
Attribute: Control (CONTROLLING, CONTROLLED, NORMAL)
Constraint: Control = CONTROLLING
  Attribute: List of References to Controlled Program Invocations
  Attribute: Program Location - String
  Attribute: Running Mode (FREE-RUN, CYCLE-LIMITED, STEP-LIMITED)
  Constraint: Running Mode = CYCLE-LIMITED
    Attribute: Remaining Cycle Count - Integer
  Constraint: Running Mode = STEP-LIMITED
    Attribute: Remaining Step Count - Integer
Constraint: Control = CONTROLLED
  Attribute: Reference to Controlling Program Invocation

6.3.2.2 Error Code

This attribute, of type integer, shall identify the last recorded error of the Program Invocation execution. A value of zero indicates that no error has been recorded. The meaning of other values is a local matter, as is the method of resetting this attribute.

6.3.2.3 Control

This attribute indicates whether (CONTROLLING) this Program Invocation is intended to be coupled to another Program Invocation as being in control, or whether (CONTROLLED) this Program Invocation, after coupling, normally receives control information from another Program Invocation (the Controlling Program Invocation). If neither case applies, this attribute shall have the value NORMAL.

When two Program Invocations are coupled in this manner, the service indications for the Start and Resume service requests received by a Controlling Program Invocation can cause state transitions in the Controlled Program Invocations which are coupled to the Controlling Program Invocation. (See 7.3.3.3 and 7.3.3.5) The result reported by a Controlling Program Invocation can contain information which reflects the effect of the service request on the Controlled Program Invocations.

The intent is to associate the execution of a task program which directs the activity of the robot with a Controlling Program Invocation, and the operation of the hardware control programs of the physical devices with Controlled Program Invocations. This allows logical asynchrony in operation between the two classes of Program Invocations, a necessary feature of robot operation. In particular, there are now three ways to stop a robot, by stopping the control program of the robot arm, by stopping the task program which has the indirect effect of stopping the arm when no movement commands are generated, and by stopping the entire system (See 7.3.3.4).

A Controlling Program Invocation can control several Controlled Program Invocations. As an example, R_ARM is a pre-defined Controlled Program Invocation. Tool management or an active auxiliary device can be modelled as another Controlled Program Invocation. Both these Program Invocation can be controlled simultaneously by the same Controlling Program Invocation which is often a user written program.

Note: It may not be required to implement Controlling and Controlled Program Invocation objects. In some applications it may be sufficient to refer to the robot arm by including the R_ARM Domain in the List of Domain References of a normal Program Invocation during the creation process. Doing so, however, precludes the ability to make the distinction between stopping the task program and stopping the robot arm motion.

6.3.2.4 List of References to Controlled Program Invocations

This attribute is present only if the Program Invocation has its Control attribute equal to CONTROLLING. It is a list of references to other Program Invocations which have their Control attribute value equal to CONTROLLED, and which have their Reference to Controlling Program Invocation attribute indicating the present Program Invocation. This list may be empty.

6.3.2.5 Program Location

This attribute, of type character string, is present only if the Program Invocation has its Control attribute equal to CONTROLLING. The use of this attribute shall be an implementation option, and if used, the format of the Program Location attribute shall be described in the PICS (See 9.4). If present, it identifies the line of source code for the Program Invocation which is currently being executed or will be executed when the Program Invocation is placed in the RUNNING state.

6.3.2.6 Running Mode

This attribute is present only if the Program Invocation has its Control attribute equal to CONTROLLING. If present, it indicates how the Program Invocation execution is governed. If the value of this attribute is FREE-RUN, the Program Invocation will stay in the RUNNING state until some local or remote event occurs which causes it to cease execution. If value of this attribute is CYCLE-LIMITED, an explicit counter is maintained containing the number of cycles to be executed. When this counter reaches zero, the Program Invocation ceases execution and returns to the IDLE state. This attribute may take the value STEP-LIMITED for implementations which support this mode. In that case, the number of steps to be executed shall be specified when the Program Invocation is started or resumed. When the number of steps has been decremented to zero, the Controlling Program Invocation shall automatically move to the STOPPED state. This mode is normally only used for debugging purposes. It shall be indicated in the PICS (See 9.4) whether the implementation supports the Running Mode STEP-LIMITED.

6.3.2.7 Remaining Cycle Count

This attribute, of type integer, is the number of cycles of the Program Invocation remaining to be executed. If the Program Invocation is not in the CYCLE-LIMITED Running Mode, this attribute is undefined.

6.3.2.8 Remaining Step Count

This attribute, of type integer, is the number of steps remaining to be executed when in Step-Limited Running Mode. This attribute is undefined unless the Program Invocation is in the STEP-LIMITED Running Mode.

6.3.2.9 Reference to Controlling Program Invocation

This attribute is present only if the value of the Control attribute is CONTROLLED. If present, it indicates the Program Invocation which acts as the Controlling Program Invocation for this Program Invocation. The referenced Program Invocation shall have its Control attribute value equal to CONTROLLING, and shall have included in its List of Controlled Program Invocation Reference attribute a reference to this Program Invocation. If no such Program Invocation exists, the value of this attribute shall be UNCONTROLLED.

6.3.3 Extensions to the Program Invocation state diagram

For Program Invocations which have their Control attribute equal to CONTROLLING, the Robot Invocation State Diagram (11.1.3 of ISO/IEC 9506-1:1990) shall be augmented. The IDLE state shall be subdivided to account for the fact that the Controlling Program Invocation may or may not be selected. The IDLE state diagram is given in Figure 10.

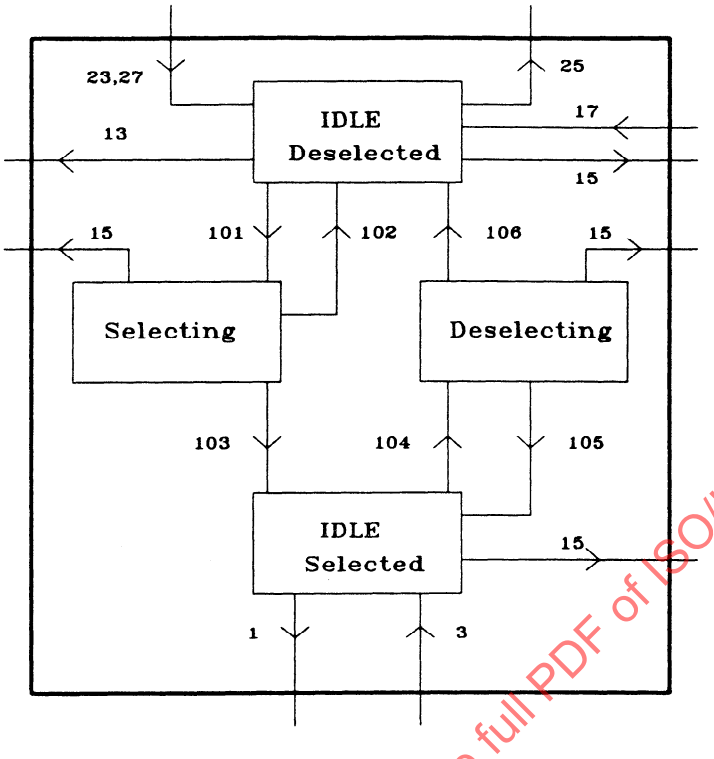

Figure 10 IDLE state diagram

The transitions numbered below 100 refer to transitions described in 11.1.3 of ISO/IEC 9506-1:1990. The additional transitions in this diagram are:

101	Select Indication
102	Select Response(-)
103	Select Response(+)
104	Select Indication, parameter = deselect
105	Select Response(-)
106	Select Response(+)

6.3.4 Robot operation states and the VMD

6.3.4.1 General

The major states of the Robot Operation Model (See clause 5) are related to the motion parameters of the robot arm(s) and the auxiliary devices which can move, and to the Controlling Program Invocation which has been selected to control the robot arm. The set of devices which are reflected in the VMD state need not include all the devices of the robot system, but it shall include the robot arm(s) and the safety interlocks. The implementor shall

supply in the PICS (See 9.4) the specific list of devices included in the VMD state. In the state descriptions which follow, the term 'robot moving' refers to motion in any of this set of devices.

The relationship between the Robot Operation State model and elements of the VMD is independent of how these states were entered; that is, this model applies even when local means are used to effect the transitions. The relationship between the Robot Operation State model of clause 5 and the state of the robot arm, auxiliary devices, and the Program Invocation which control them is as follows:

6.3.4.2 ROBOT-IDLE state

The robot has no Controlling Program Invocation in the VMD. The VMD attribute 'Reference to Selected Program Invocation' has the value NONE. The R_ARM Program Invocation has its Reference to Controlling Program Invocation attribute equal to UNCONTROLLED. Otherwise, the VMD is fully operational.

6.3.4.3 ROBOT-LOADED state

The robot has one or more Controlling Program Invocations in the VMD. The VMD attribute Reference to Selected Program Invocation has the value NONE. The R_ARM Program Invocation has its Reference to Controlling Program Invocation attribute equal to UNCONTROLLED.

6.3.4.4 ROBOT-READY state

The VMD attribute Reference to Selected Program Invocation has the value of a reference to a Controlling Program Invocation. The R_ARM Program Invocation Reference to Controlling Program Invocation attribute references this same Program Invocation which is in the IDLE state.

6.3.4.5 ROBOT-EXECUTING state

The robot has been placed in operation; motion is enabled and the arm may be moving. The Program Invocation referenced by the VMD attribute Reference to Selected Program Invocation is in the RUNNING or the STOPPED state.

6.3.4.6 ROBOT-MOTION-PAUSED state

The robot has ceased motion. The Program Invocation referenced by the VMD attribute Reference to Selected Program Invocation is in the RUNNING or the STOPPED state.

6.3.4.7 MANUAL-INTERVENTION-REQUIRED state

In this state, the robot has ceased motion. The VMD Logical Status is NO-STATE-CHANGES-ALLOWED, LIMITED-SERVICES-PERMITTED or SUPPORT-SERVICES-ALLOWED. The VMD Physical Status is NEEDS-COMMISSIONING. The state of the other attributes of the VMD is a local matter.

6.3.5 Description of Robot Operation state transitions

6.3.5.1 General

This subclause describes the sequence of MMS services which correspond to the transitions of the Robot Operation State Model (See Figure 8). The relationship between the MMS services and the Robot Operation State Model depends on whether the Selected Program Invocation (the Program Invocation referenced by the VMD attribute) is created dynamically through MMS CreateProgramInvocation service, or is preexisting and predefined in the robot system controller. The transitions can be accomplished by MMS service procedures as follows:

6.3.5.2 Load - 1

If the Robot does not contain a predefined Program Invocation which has its Control attribute value equal to CONTROLLING, the Robot will move from the ROBOT-IDLE state to ROBOT-LOADED state by the creation of a Program Invocation whose Control attribute value is equal to CONTROLLING. The MMS service employed is CreateProgramInvocation. If the Robot contains a predefined Program Invocation which has its Control attribute value equal to CONTROLLING, the Robot will pass directly through this state at power up to the ROBOT-LOADED state and this transition will not occur.

In the latter case, the Robot may pass directly to the ROBOT-READY state. Note also that while the creation of such a Program Invocation can be preceded by the creation and loading of Domains, that process is not reflected in the Robot Operation State diagram.

6.3.5.3 Clear - 2

The Robot moves from ROBOT-LOADED state either to the ROBOT-LOADED state or to the ROBOT-IDLE state by the deletion of a Controlling Program Invocation using the DeleteProgramInvocation service. If there are multiple Controlling Program Invocations present, deleting each one will leave the Robot in the ROBOT-LOADED state until the last one is deleted. Deletion of the last such Program Invocation will return the Robot to ROBOT-IDLE. The VMDReset service can also cause this transition.

This transition can only occur if a previous Load transition has occurred. Note also that subsequent deletion of the Domains which were constituents of the Program Invocation has no effect on the Robot Operation State diagram.

6.3.5.4 Assign - 3

If the Robot does not contain a predefined Program Invocation which is a Controlling Program Invocation and which is referenced by the Reference to Selected Program Invocation attribute of the VMD, the Robot will move from the ROBOT-LOADED state to the ROBOT-READY state by the process of selecting a Program Invocation for control of the robot. The Program Invocation selected shall have its Control attribute value equal to CONTROLLING. The MMS service used is the Select service. If a predefined Program Invocation exists which is referenced by the Reference to Selected Program Invocation attribute of the VMD, the Robot will move directly to the ROBOT-READY state at power up and this transition will not occur.

6.3.5.5 Deassign - 4

The Robot is moved from the ROBOT-READY state to the ROBOT-LOADED state by deselecting the Controlling Program Invocation which is referenced by the Reference to Selected Program Invocation attribute of the VMD. The MMS service used is the Select service with a null parameter specified. See 7.4.4.

This transition can occur only if an Assign transition has previously occurred.

6.3.5.6 Program Start - 5

The Robot is moved from the ROBOT-READY state to the ROBOT-EXECUTING state by starting the Controlling Program Invocation which is referenced by the Reference to Selected Program Invocation attribute of the VMD. The MMS service used is the Start service.

6.3.5.7 End of Execution- 6

The Robot is moved from ROBOT-EXECUTING state to the ROBOT-LOADED state through the normal programmed operation of the Program Invocation.

6.3.5.8 Program Reset - 7

The Robot can be moved from the ROBOT-MOTION-PAUSED state to the ROBOT-READY state by resetting the Controlling Program Invocation which is referenced by the Reference to Selected Program Invocation attribute of the VMD. The MMS service used is Reset.

6.3.5.9 Pause - 8

The robot is moved from ROBOT-EXECUTING to ROBOT-MOTION-PAUSED state by disabling motion, i.e. setting motion enabled equal to false. This is normally accomplished by stopping the Controlled Program Invocation R_ARM. The MMS service used is Stop.

6.3.5.10 Continue - 9

The robot is moved from ROBOT-MOTION-PAUSED state to ROBOT-EXECUTING state by resuming the Controlling Program Invocation which is referenced by the Reference to Selected Program invocation attribute of the VMD. The MMS service used is Resume.

6.3.5.11 Clear(Abort) - 10

The transition of the Robot from ROBOT-MOTION-PAUSED state to ROBOT-IDLE state is the same as Clear except that the Controlling Program Invocation has not completed the most recent cycle. The MMS service used is the robot specific VMDReset Service.

6.3.5.12 Clear(Abandon) - 11

The transition of the Robot from ROBOT-READY state to ROBOT-IDLE state is the same as Clear except that the Controlling Program Invocation has not completed the most recent cycle. The MMS service used is the robot specific VMDReset service.

6.3.5.13 Emergency action - 12

Through some local action, such as activation of the emergency stop circuit, the robot may move from any state to the MANUAL-INTERVENTION-REQUIRED state. Any Program Invocations which are in the RUNNING state may be removed from that state. However, the resulting state of these Program Invocations is a local matter. This transition can also be accomplished through the robot specific VMDStop service.

6.3.6 Calibration procedure

The calibrated state of a physical resource is modeled as an attribute of the Domain which models the physical resource. There are three possible values for this attribute, NOT-CALIBRATED, CALIBRATING, and CALIBRATED. A predefined Domain containing the calibrate function and a predefined Program Invocation using that Domain exist to perform calibration of the physical resource. While the R_CAL Program Invocation is executing, the Device Calibrated attribute of the Domain representing the physical resource shall be set to CALIBRATING. Upon completion, the R_CAL Program Invocation shall be set to the IDLE state. If the calibrate function was successful, the Device Calibrated attribute shall be set to CALIBRATED, otherwise to NOT-CALIBRATED. If all such physical resources have this attribute equal to CALIBRATED, then the VMD attribute All Physical Resources Calibrated shall have the value TRUE.

The R_CAL Program Invocation differs from Controlling Program Invocations in that it is a normal Program Invocation; its presence shall not cause the robot state to be ROBOT-LOADED.

6.3.7 Subsystem activation and deactivation

Activation and deactivation of auxiliary devices can be modeled as write operations onto predefined variables within the appropriate predefined Domains or by state transitions of a predefined Program Invocation. Specifically, the application of power to the robot arm can be modeled as a boolean variable within the predefined Domain for the robot arm. Clause 8 provides guidance on the selection of names for such variables.

6.3.8 Status reporting

6.3.8.1 VMD object status

The value of attributes of the VMD shall be obtained by using the MMS Status service.

6.3.8.2 Robot arm object status

The status of the attributes of the robot arm that can be obtained through MMS shall be obtained by using the MMS Read service. For each attribute value that can be accessed a standardized Named Variable object has been defined. Using the MMS Read service to read this Named Variable object shall provide the status of the associated attribute.

6.3.8.3 Controlling Program Invocation object status

Additional attributes of the MMS Program Invocation object have been defined in this standard. The value or status of these attributes shall be obtained by using the MMS GetProgramInvocationAttributes service.

6.3.9 Spontaneous status reporting

The status of objects within the robot VMD can also be spontaneously reported. This can occur through the use of the MMS UnsolicitedStatus service or through the use of MMS EventNotification service.

Standardized names are defined for various Event Action objects and Event Condition objects. These standardized names are intended for, but not limited to, use with the EventNotification service to report status. Event notifications may be generated for conditions other than those with standardized names. It is not specified in this part of ISO/IEC 9506 when the UnsolicitedStatus and EventNotification services should be used.

6.4 Definition of robot-specific objects which map to other MMS abstract objects

This part of ISO/IEC 9506 does not define any robot-specific objects which map to other MMS abstract objects.

6.5 Definition of new MMS abstract objects to support other robot-specific objects

This part of ISO/IEC 9506 does not define any additional MMS abstract objects.

7 Robot specific services and protocol

7.1 Robot application context definition

For the purpose of being able to use an application which only contains the ACSE and MMS as ASEs, this part of ISO/IEC 9506 uses the object identifier value and the object descriptor value defined in 17.12 of ISO/IEC 9506-2:1990.

7.2 Robot specific abstract syntax definition

This part of ISO/IEC 9506 assigns the ASN.1 object identifier value

```
{iso standard 9506 part(3) mms-robot-syntax-version1(1)}
```

to the abstract syntax defined in this clause.

7.3 Use of MMS services

7.3.1 Robot specific ASN.1 module definition

The MMS services and protocol were developed to be used by a wide range of manufacturing devices. This clause defines the robot services and protocol for those elements identified as requiring companion standard definition in ISO/IEC 9506-2. These definitions shall be used when the abstract syntax defined in this part of ISO/IEC 9506 is negotiated. This clause further clarifies how MMS services are to be used with robots.

All ASN.1 definitions provided in this part of ISO/IEC 9506 are part of the ASN.1 Module "ISO-9506-MMS-ROBOT-1". The beginning and closing statements indicating that each ASN.1 definition provided is a part of this module is omitted in order to make reading of the document easier. Each ASN.1 definition provided implicitly contains the statement:

```
ISO-9506-MMS-ROBOT-1
  {iso standard 9506 part(3) mms-robot-module-version1(2)}
DEFINITIONS ::= BEGIN
```

at the beginning of the definition and contains the keyword "END" at the end of the definition.

Note: ISO-9506-MMS-ROBOT-1 represents revision number 1 of the MMS (ISO/IEC 9506-3) Robot Companion Standard.

Many of the terms and abbreviations used in this clause use the terminology of MMS service and protocol descriptions (see clause 5 of ISO/IEC 9506-1:1990 and clause 5 of ISO/IEC 9506-2:1990). In particular, refer to clause 5 of ISO/IEC 9506-1:1990 for general rules on how to interpret the service tables in this part of ISO/IEC 9506.

```
IMPORTS   MMSpdu,
          ParameterSupportOptions,
          ServiceSupportOptions,
          Identifier,
          Integer16,
          StatusResponse
    FROM MMS-General-Module-1
    {iso standard 9506 part(2) mms-general-module-version1(2) };
```

7.3.2 VMD support services and protocol

7.3.2.1 Status and UnsolicitedStatus service

The MMS Status service provides a way for the responding robot to indicate the generic status of its VMD. The UnsolicitedStatus service provides a way for the robot to indicate to a host the generic status of its VMD without having a status service request. It is a local matter when UnsolicitedStatus shall be sent.

Both services allow for companion standard specific information to be included in the request and in the response.

This part of ISO/IEC 9506 does not prescribe any additional parameters to be added to a Status Request.

The Robot Status Detail parameter shall be used in the response and confirm primitives of the Status service and in the request and indication primitives of the UnsolicitedStatus service.

7.3.2.1.1 Robot Status Detail parameter

The structure of the Robot Status Detail parameter is shown in Table 2.

Table 2 Robot status detail

Parameter Name	Req Rsp	Ind Cnf
Robot VMD State	M	M(=)
Robot Specific Status	M	M(=)
Robot Specific Status Mask	U	U(=)
Selected Program Invocation	M	M(=)

7.3.2.1.1.1 Robot VMD State

This parameter, of type integer, shall convey the value of the Robot VMD State attribute of the VMD. The Robot VMD State attribute is defined in 6.3.4 and described in 5.3.1.3.

7.3.2.1.1.2 Robot Specific Status

This parameter, of type bitstring, shall convey the boolean values of the set of robot specific attributes:

Safety Interlocks Violated,
Any Physical Resource Power On,
All Physical Resources Calibrated,
Local Control

These attributes are defined in 6.1.3.2. Local Control is described in 5.3.1.4.

7.3.2.1.1.3 Robot Specific Status Mask

This optional parameter, of type bitstring, shall convey the significance of the corresponding bit in the Robot Specific Status parameter. If a bit in this bitstring is one, the corresponding bit in the Robot Specific Status parameter is significant. If a bit in this bitstring is zero, the corresponding bit in the Robot Specific Status parameter shall be ignored. The default value for this parameter is all one's.

7.3.2.1.1.4 Selected Program Invocation

This parameter indicates the Program Invocation which has been selected to be the Controlling Program Invocation for the Robot System. If no Program Invocation has been selected, this parameter shall have the value NONE.

7.3.2.1.2 Robot Specific Status protocol

The abstract syntax of the CS-Status-Request, the CS-Status-Response, and the CS-Unconfirmed-Status is specified below and described in the paragraphs that follow. Clause 5.5 of ISO/IEC 9506-2 describes the derivation of all parameters for which explicit derivations are not provided in this clause.

```
CS-Status-Request ::= NULL

CS-Status-Response ::= RobotStatusDetail

CS-UnsolicitedStatus ::= RobotStatusDetail

RobotStatusDetail ::= SEQUENCE {
   robotVMDState            [0] IMPLICIT RobotVMDState,
   robotSpecificStatus      [1] IMPLICIT RobotSpecificStatus,
   robotSpecificStatusMask [2] IMPLICIT RobotSpecificStatus DEFAULT '11111'B,
   selectedProgramInvocation  CHOICE {
   programInvocation             [3] IMPLICIT Identifier,
   noneSelected                  [4] IMPLICIT NULL
   }
}
RobotVMDState ::= INTEGER {
   robot-idle                (0),
   robot-loaded              (1),
   robot-ready               (2),
   robot-executing           (3),
   robot-motion-paused       (4),
   manualInterventionRequired (5)
   }

RobotSpecificStatus ::= BITSTRING {
   safetyInterlocksViolated          (0),
   anyPhysicalResourcePowerOn        (1),
   allPhysicalResourcesCalibrated    (2),
   localControl                      (3)
   }
```


7.3.3 Program Invocation management services and protocol

7.3.3.1 General structure

This subclause describes the services and protocol for the robot specific usage of the Program Invocation management operations. The model of a Program Invocation is enlarged to allow the concept of interworking of two related Program Invocations. In such a pair of interworking Program Invocations, one Program Invocation is referred to as the Controlling Program Invocation, the other as the Controlled Program Invocation. In general, the two Program Invocations are modeled as having an inter-process communication channel available to them over which they pass messages which provide both data and synchronization of the processes. However, for the most part, such messages are not evident to the remote MMS user except insofar as they cause the state of the Program Invocation to be altered.

The model of Program Invocation behaviour is extended so that when a Start or Resume indication is received by a Controlling Program Invocation, as part of its service procedure, it shall cause a transition within the related Controlled Program Invocation. In its response to the service request, a Controlling Program Invocation shall incorporate into its response information appropriate to the Controlled Program Invocation. The service procedure for these services directed at a Controlled Program Invocation are normal in that they do not propagate to other Program Invocations.

The relationship between Program Invocations is many to one; a Controlling Program Invocation can control many Controlled Program Invocations; a Controlled Program Invocation can be controlled by only one Controlling Program Invocation at any specific time. See also the Select service in 7.4.4.

7.3.3.2 CreateProgramInvocation service

The CreateProgramInvocation service request shall be extended to allow the specification of the Control attribute of the Program Invocation being created. This shall be accomplished by adding a parameter to the CreateProgramInvocation Request.

7.3.3.2.1 CS-CreateProgramInvocation-Request

The structure of the CS-CreateProgramInvocation-Request parameter is shown in Table 3.

Table 3 CS-CreateProgramInvocation-Request parameter

Parameter Name	Req	Ind
Control	M	M(=)

7.3.3.2.1.1 Control

This parameter shall indicate the value of the Control attribute of the Program Invocation. This parameter may have the value CONTROLLING, CONTROLLED, or NORMAL.

7.3.3.2.2 CS-CreateProgramInvocation-Response

This part of ISO/IEC 9506 defines no additional parameters for the CreateProgramInvocation-Response.

7.3.3.2.3 Extended service procedure

The service procedure specified in 11.2.2 of ISO/IEC 9506-1:1990 shall be performed. If that procedure completes successfully, the Control attribute shall be set in accordance with the value of the Control parameter in the service request and the value of the Error Code attribute shall be set to zero.

a) If the Control parameter in the service request is CONTROLLING, the attribute value of the List of References to Controlled Program Invocations shall be set to an empty list. The Program Location attribute shall be set to an empty string and the Running Mode attribute shall be set to FREE-RUN.

b) If the Control parameter in the service request is CONTROLLED, the Reference to Controlling Program Invocation attribute shall be set to UNCONTROLLED.

7.3.3.2.4 CreateProgramInvocation protocol

The abstract syntax of the CS-CreateProgramInvocation-Request is specified below and described in the paragraphs that follow. Clause 5.5 of ISO/IEC 9506-2:1990 describes the derivations of all parameters for which explicit derivations are not provided in this clause.

```
CS-CreateProgramInvocation-Request  ::= INTEGER {
        normal      (0),
        controlling (1),
        controlled  (2)
        }

CS-CreateProgramInvocation-Response ::= NULL
```

7.3.3.3 Start service

The Start service allows a client to request a responding robot to initiate execution of a Program Invocation. The service procedure of the Start service is extended through the use of the CS-Start-Request parameter. The service procedure is further conditioned on whether the Control attribute of the Program Invocation is CONTROLLING, CONTROLLED or NORMAL and on whether the client has control of the R_CTRL semaphore.

7.3.3.3.1 CS-Start-Request

The structure of the CS-Start-Request parameter is shown in Table 4. The CS-Start-Request parameter shall be non null if and only if the Control attribute of the Program Invocation has the value CONTROLLING.

Table 4 CS-Start-Request parameter

Parameter Name	Req	Ind
Start Location	U	U(=)
Running Mode	C	C(=)
No Limit	S	S(=)
Cycle Count	S	S(=)
Step Count	S	S(=)

7.3.3.3.1.1 Start Location

This optional parameter, of type character string, shall be present only if the Program Invocation has its Control attribute value equal to CONTROLLING. If the Control attribute value is equal to CONTROLLING, the use of this parameter shall be a user option. If this parameter is implemented, its format shall be described in the PICS (See 9.4). The value of this parameter shall indicate the starting location within the Program Invocation at which to begin execution. If the Start Location parameter is omitted from the service request, execution shall begin at the first step of the program. The meaning of the first program step or a default first step is a local matter. This meaning shall be defined for any implementation in the PICS (See 9.4).

7.3.3.3.1.2 Running Mode

This parameter shall be present only if the Control attribute value of the Program Invocation is equal to CONTROLLING. The value of this parameter shall indicate the value for the Running Mode attribute of the Program Invocation. Depending on which Running Mode of the Program Invocation is selected, one of the following parameters shall be present.

7.3.3.3.1.3 No Limit

This parameter, of type null, shall be chosen if the value of the Running Mode attribute is to be set to FREE-RUN.

7.3.3.3.1.4 Cycle Count

This parameter, of type integer, shall be chosen if the value of the Running Mode attribute is to be set to CYCLE-LIMITED, and the value of the Remaining Cycle Count attribute shall be set to this parameter value. The value of this parameter shall be greater than zero.

7.3.3.3.1.5 Step Count

This parameter, of type integer, shall be chosen if the value of the Running Mode attribute is to be set to STEP-LIMITED, and the value of the Remaining Step Count attribute shall be set to this parameter value. The value of this parameter shall be greater than zero. Implementation of the STEP-LIMITED Running Mode shall be defined in the PICS (see 9.4). If not supported, this choice may not be selected.

7.3.3.3.2 CS-Start-Response

This part of ISO/IEC 9506 defines no additional parameters for the Start-Response.

7.3.3.3.3 Extended service procedure

The following service procedure shall apply to the Start service.

a) The error checks of the service procedure in 11.4.2 of ISO/IEC 9506-1:1990 shall be performed.

b) If the requesting MMS user does not own the R_CTRL semaphore, a Result(-) shall be returned.

c) If the Control attribute of the Program Invocation identified by the Program Invocation Name parameter of the Start service request does not have the value CONTROLLING, verify that the Start Location and Running Mode parameters are not present. If either is present, return a Result(-). Otherwise, the remainder of the Start service procedure of 11.4.2 of ISO/IEC 9506-1:1990 shall be performed and the rest of this procedure shall be skipped.

d) If the Program Invocation identified by the Program Invocation Name parameter of the Start service request has its Control attribute equal to CONTROLLING, verify that the Program Invocation is referenced by the Reference to Selected Program Invocation attribute of the VMD. If this is not the case, return a Result(-) and skip the remainder of this procedure. Otherwise, the following steps shall be performed for each element on the List of References to Controlled Program Invocations attribute of this Program Invocation:

   1) Verify that the referenced Program Invocation has its Control attribute value equal to CONTROLLED and that its Reference to Controlling Program Invocation attribute references this Program Invocation.

   2) If the referenced Controlled Program Invocation is in the IDLE state, perform a Start procedure for this referenced Controlled Program Invocation and move it into the RUNNING state.

   3) If the referenced Controlled Program Invocation is in the STOPPED state, perform a Resume procedure for this referenced Controlled Program Invocation and move it into the RUNNING state.

4) If the Controlled Program Invocation cannot be placed in the RUNNING state, then for every previous Controlled Program Invocation of the list, perform a Stop procedure. If any such Stop procedure fails, it is a local matter how to treat this situation. A VMDStop procedure may be the appropriate action. Finally, return a Result(-) for this service request and skip the remainder of this procedure.

e) If the Program Invocation identified by the Program Invocation Name parameter of the Start service request has its Control attribute equal to CONTROLLING, the attributes of the Controlling Program Invocation shall be set as follows:

1) If the No Limit parameter is selected in the service request, the Running Mode attribute of the Program Invocation shall be set to FREE-RUN.

2) If the Cycle Count parameter is selected in the service request, the Running Mode attribute of the Program Invocation shall be set to CYCLE-LIMITED and the Remaining Cycle Count attribute of the Program Invocation shall be set to the value of the Cycle Count parameter.

3) If the Step Count parameter is selected in the service request, the Running Mode attribute of the Program Invocation shall be set to STEP-LIMITED and the Remaining Step Count attribute of the Program Invocation shall be set to the value of the Step Count parameter.

4) If the Start Location parameter is present in the service request, the value of this parameter shall condition the program control information of the Program Invocation in order to provide a starting point for the execution of the Program Invocation. The representation used to convey the starting location is a local matter, and in general will depend on the programming language used. The format used for the Start Location parameter shall be described in the PICS (See 9.4).

5) If the Start Location parameter is absent from the service request, the default value of the starting location shall be used in the execution of the Program Invocation.

f) If the Program Invocation identified by the Program Invocation Name parameter of the Start service request has its Control attribute equal to CONTROLLED, and the Program Invocation is bound to a Domain which represents a robot arm, set the Motion Enabled attribute of the Domain to TRUE.

g) The remainder of the Start service procedure of 11.4.2 of ISO/IEC 9506-1:1990 shall be performed.

7.3.3.3.4 Start protocol

The abstract syntax of the CS-Start-Request is specified below and described in the paragraphs that follow. Clause 5.5 of ISO/IEC 9506-2:1990 describes the derivation of all parameters for which explicit derivations are not provided in this clause.

```
CS-Start-Request  ::= [0] CHOICE {
   normal         NULL,
   controlling    SEQUENCE {
     startLocation [0] IMPLICIT VisibleString OPTIONAL,
     startCount    [1] StartCount DEFAULT cycleCount 1
} }
```

```
StartCount ::= CHOICE {
  noLimit           [0] IMPLICIT NULL,
  cycleCount        [1] IMPLICIT INTEGER,
  stepCount         [2] IMPLICIT INTEGER
}

CS-Start-Response ::= NULL
```

Note: Because of the convention of the comment field in the ConfirmedRequestPDU production in 7.1 of ISO/IEC 9506-2:1990, if the normal choice of the CS-Start-Request is selected, the CS-Start-Request will not be transmitted.

7.3.3.3.4.1 Controlling Program Invocation

If the Control attribute of the Program Invocation has the value CONTROLLING, the controlling choice shall be made in the CS-Start-Request; if the Control attribute is CONTROLLED or NORMAL, the normal choice shall be made.

7.3.3.3.4.2 Running Mode

The value of the Running Mode parameter shall be inferred by the choice made within the StartCount type. The choice of nolimit shall indicate that the Running Mode shall be set to FREE-RUN. The choice of cycleCount shall indicate that the Running Mode shall be set to CYCLE-LIMITED and the Remaining Cycle Count attribute shall be set to the value indicated. The choice of stepCount shall indicate that the Running Mode shall be set to STEP-LIMITED and the Remaining Step Count attribute shall be set to the value indicated.

For systems which do not support the STEP-LIMITED Running Mode, the StartCount choice shall be limited to values nolimit ([0]) or cycleCount ([1]).

7.3.3.4 Stop service

This part of ISO/IEC 9506 defines no additional parameters for the Stop-Request of Stop-Response.

7.3.3.4.1 Extended service procedure

The following service procedure shall apply to the Stop service.

a) The error checks of the Stop service procedure of 11.5.2 of ISO/IEC 9506-1:1990 shall be performed.

b) If the requesting MMS user does not own the R_CTRL semaphore, a Result(-) shall be returned.

c) If the Control attribute of the Program Invocation is CONTROLLED and the Program Invocation is bound to a Domain which represents a robot arm, set the Motion Enabled attribute to FALSE.

d) The remainder of the Stop service procedure of 11.5.2 of ISO/IEC 9506-1:1990 shall be performed.

Stopping a Robot can be done in one of two ways using the Stop service. If the Controlling Program Invocation is stopped, the task program controlling the operation of the robot will be stopped. At some later time, after the robot has completed the last directive issued by the Controlling Program Invocation, robot motion will cease, even though the Controlled Program Invocation representing the robot arm control program remains in the RUNNING state. If the Controlled Program Invocation which represents the robot arm control program is stopped, action of the robot arm ceases immediately, i.e. before the last programmed step has been completed but in such a way as to maintain the last programmed path. A third method of stopping the robot is provided by the VMDStop service (See 7.4.2).

7.3.3.4.2 Stop protocol

This part of ISO/IEC 9506 does not define any syntax extensions for the Stop protocol specified in ISO/IEC 9506-2.

```
CS-Stop-Request ::= NULL

CS-Stop-Response ::= NULL
```

7.3.3.5 Resume service

The Resume service allows a responding robot to resume execution of a Program Invocation from a point where execution was previously stopped.

7.3.3.5.1 CS-Resume-Request

The structure of the CS-Resume-Request parameter is shown in Table 5.

Table 5 CS-Resume-Request parameter

Parameter Name	Req	Ind
Resume Type	C	C(=)
Continue Mode	S	S(=)
Change Mode	S	S(=)
No Limit	S	S(=)
Cycle Count	S	S(=)
Step Count	S	S(=)

7.3.3.5.1.1 Resume Type

This parameter shall be present if and only if the Control attribute of the Program Invocation is CONTROLLING. If present, it indicates whether the Program Invocation shall resume executing with the Running Mode attribute of the Program Invocation changed or unchanged.

7.3.3.5.1.2 Continue Mode

This parameter shall be selected if the Program Invocation is to resume execution with the Running Mode attribute unchanged.

7.3.3.5.1.3 Change Mode

This parameter shall be selected if the Program Invocation is to resume execution with the Running Mode attribute changed to a new value. If this parameter is selected then one of the following parameters shall also be present.

7.3.3.5.1.4 No Limit

This parameter, of type null, shall be chosen if the value of the Running Mode attribute is to be changed to FREE-RUN.

7.3.3.5.1.5 Cycle Count

This parameter, of type integer, shall be chosen if the value of the Running Mode attribute is to be changed to CYCLE-LIMITED, and the value of the Remaining Cycle Count attribute shall be set to this parameter value. The value of this parameter shall be greater than zero.

7.3.3.5.1.6 Step Count

This parameter, of type integer, shall be chosen if the value of the Running Mode attribute is to be changed to STEP-LIMITED, and the value of the Remaining Step Count attribute shall be set to this parameter value. The value of this parameter shall be greater than zero. Implementation of the STEP-LIMITED Running Mode shall be defined in the PICS (See 9.4). If not supported, this choice may not be selected.

7.3.3.5.2 CS-Resume-Response

This part of ISO/IEC 9506 defines no additional parameters for the Resume-Response.

7.3.3.5.3 Extended service procedure

The following service procedure shall apply to the Resume service.

a) The error checks of the service procedure of 11.6.2 of ISO/IEC 9506-1:1990 shall be performed.

b) If the requesting MMS user does not own the R_CTRL semaphore, a Result(-) shall be returned.

c) If the Control attribute of the Program Invocation does not have the value CONTROLLING, verify that the Resume Type parameter is not present. If it is present, return a Result(-). The remainder of the Resume service procedure of 11.6.2 of ISO/IEC 9506-1:1990 shall be performed and the rest of this procedure shall be skipped.

d) If the Program Invocation identified by the Program Invocation Name parameter of the Resume service request has its Control attribute equal to CONTROLLING, the following steps shall be performed for each element on the List of References of Controlled Program Invocation attribute of this Program Invocation:

   1) Verify that the referenced Program Invocation has its Control attribute value equal to CONTROLLED and that its Reference to Controlling Program Invocation attribute references this Program Invocation.

   2) If the referenced Controlled Program Invocation is in the IDLE state, perform a Start procedure for the referenced Controlled Program Invocation and move it into the RUNNING state.

   3) If the referenced Controlled Program Invocation is in the STOPPED state, perform a Resume procedure for the referenced Controlled Program Invocation and move it into the RUNNING state.

   4) If the Controlled Program Invocation cannot be placed in the RUNNING state, then for every previous Controlled Program Invocation of the list, perform a Stop procedure. If any such Stop procedure fails, it is a local matter how to treat this situation. A VMDStop procedure may be the appropriate action. Finally, return a Result(-) for this service request and skip the remainder of this procedure.

e) If Continue Mode is selected perform the following check. If the value of the Running Mode attribute of the Program Invocation is CYCLE-LIMITED and the value of the Remaining Cycle Count attribute is zero, or if the value of the Running Mode attribute is STEP-LIMITED and the value of the Remaining Step Count attribute is zero, a Result(-) shall be returned. Otherwise, the Resume service procedure described of 11.6.2 of ISO/IEC 9506-1:1990 shall be performed.

f) If the Change Mode parameter is selected in the Resume Type parameter, the attributes of the Program Invocation shall be set as follows:

   1) If the No Limit parameter is selected in the service request, the Running Mode attribute of the Program Invocation shall be set to FREE-RUN.

   2) If the Cycle Count parameter is selected in the service request, the Running Mode attribute of the Program Invocation shall be set to CYCLE-LIMITED and the Remaining Cycle Count attribute of the Program Invocation shall be set to the value of the Cycle Count parameter.

3) If the Step Count parameter is selected in the service request, the Running Mode attribute of the Program Invocation shall be set to STEP-LIMITED and the Remaining Step Count attribute of the Program Invocation shall be set to the value of the Step Count parameter.

g) The remainder of the Resume service procedure of 11.6.2 of ISO/IEC 9506-1:1990 shall be performed.

7.3.3.5.4 Resume protocol

The abstract syntax of the CS-Resume-Request is specified below and described in the paragraphs that follow. Clause 5.5 of ISO/IEC 9506-2:1990 describes the derivation of all parameters for which explicit derivations are not provided in this clause.

```
CS-Resume-Request ::= [0] CHOICE {
 normal          NULL,
 controlling     SEQUENCE { CHOICE {
   continueMode        [0] IMPLICIT NULL,
   changeMode          [1] StartCount
   } } }

CS-Resume-Response ::= NULL
```

Note: Because of the convention of the comment field in the ConfirmedRequestPDU production in 7.1 of ISO/IEC 9506-2:1990, if the normal choice of the CS-Resume-Request is selected, the CS-Resume-Request will not be transmitted.

7.3.3.5.4.1 Controlling Program Invocation

If the Control attribute of the Program Invocation has the value CONTROLLING, the controlling choice shall be made in the CS-Resume-Request; if the Control attribute is CONTROLLED or NORMAL, the normal choice shall be made.

7.3.3.5.4.2 Resume Type

The value of Resume Type parameter shall be inferred from the choice made within CS-Resume-Request. If the continueMode choice is made, the value of the Resume Type parameter shall be Continue Mode. If the changeMode choice is made, the value of the Resume Type parameter shall be Change Mode.

7.3.3.5.4.3 Change Mode

The value of the Change Mode parameter shall be inferred by the choice made within the StartCount type. The choice of noLimit shall indicate that the Running Mode shall be set to FREE-RUN. The choice cycleCount shall indicate that the Running Mode shall be set to CYCLE-LIMITED, and the Remaining Cycle Count attribute shall be

set to the value of this parameter. The choice stepCount shall indicate that the Running Mode shall be set to STEP-LIMITED, and the Remaining Step Count attribute shall be set to the value of this parameter.

7.3.3.6 Reset service

This part of ISO/IEC 9506 defines no additional parameters for the Reset-Request or Reset-Response.

7.3.3.6.1 Extended service procedure

The following service procedure shall apply to the Reset service.

a) The error checks of the service procedure of 11.7.2 of ISO/IEC 9506-1:1990 shall be performed.

b) If the requesting MMS user does not own the R_CTRL semaphore, a Result(-) shall be returned.

c) The remainder of the service procedure of 11.7.2 of ISO/IEC 9506-1:1990 shall be performed.

7.3.3.6.2 Reset protocol

This part of ISO/IEC 9506 does not define any syntax extensions for the Reset protocol specified in ISO/IEC 9506-2.

```
CS-Reset-Request ::= NULL

CS-Reset-Response ::= NULL
```

7.3.3.7 Kill service

This part of ISO/IEC 9506 defines no additional parameters for Kill-Request or Kill-Response.

7.3.3.7.1 Extended service procedure

The following service procedure shall apply to the Kill service.

a) The error checks of the service procedure of 11.8.2 of ISO/IEC 9506-1:1990 shall be performed.

b) If the requesting MMS user does not own the R_CTRL semaphore, a Result(-) shall be returned.

c) The remainder of the service procedure of 11.8.2 of ISO/IEC 9506-1:1990 shall be performed.

7.3.3.7.2 Kill protocol

This part of ISO/IEC 9506 does not define any syntax extensions for the Kill protocol specified in ISO/IEC 9506-2.

```
CS-Kill-Request  ::= NULL

CS-Kill-Response ::= NULL
```

7.3.3.8 DeleteProgramInvocation service

This part of ISO/IEC 9506 defines no additional parameters for the DeleteProgramInvocation-Request or the DeleteProgramInvocation-Response.

7.3.3.8.1 Extended service procedure

The following service procedure shall apply to the DeleteProgramInvocation service.

a) The error checks of the service procedure of 11.3.2 of ISO/IEC 9506-1:1990 shall be performed.

b) If the requesting MMS user does not own the R_CTRL semaphore, a Result(-) shall be returned.

c) If the Control attribute of the Program Invocation to be deleted has the value CONTROLLED, and the value of the Reference to Controlling Program Invocation attribute is not UNCONTROLLED, then verify that the Program Invocation referenced by the Reference to Controlling Program Invocation attribute is in the IDLE state. If this is not true, return a Result(-) and skip the remainder of this procedure. Otherwise alter the List of References to Controlled Program Invocations attribute of the Program Invocation referenced by the Reference to Controlling Program Invocation attribute by removing the reference to the Program Invocation about to be deleted.

d) If the Control attribute of the Program Invocation to be deleted has the value CONTROLLING, then for each Program Invocation which appears on the List of References to Controlled Program Invocations, replace the value of the Reference to Controlling Program Invocation attribute with UNCONTROLLED.

e) The remainder of the service procedure of 11.3.2 of ISO/IEC 9506-1:1990 shall be performed.

7.3.3.8.2 DeleteProgramInvocation protocol

This part of ISO/IEC 9506 does not define any syntax extensions for the DeleteProgramInvocation protocol specified in ISO/IEC 9506-2.

```
CS-DeleteProgramInvocation-Request ::= NULL

CS-DeleteProgramInvocation-Response ::= NULL
```

7.3.3.9 GetProgramInvocationAttributes service

The GetProgramInvocationAttributes service is used to request that a responding robot return all of the attributes associated with a specified Program Invocation including all robot specific Program Invocation attributes.

7.3.3.9.1 CS-GetProgramInvocationAttributes-Request

This part of ISO/IEC 9506 defines no additional parameters for the GetProgramInvocationAttributes-Request.

7.3.3.9.2 CS-GetProgramInvocationAttributes-Response

The structure of the CS-GetProgramInvocationAttributes-Response parameter is shown in Table 6.

Table 6 CS-GetProgramInvocationAttributes-Response parameter

Parameter Name	Rsp	Cnf
Error Code	M	M(=)
Control	M	M(=)
Controlling	S	S(=)
List of Program Invocations	M	M(=)
Program Location	C	C(=)
Running Mode	M	M(=)
Free Running	S	S(=)
Remaining Cycle Count	S	S(=)
Remaining Step Count	S	S(=)
Controlled	S	S(=)
Controlling Program Invocation	C	C(=)
Normal	S	S(=)

7.3.3.9.2.1 Error Code

This parameter, of type integer, shall contain the Error Code attribute of the Program Invocation.

7.3.3.9.2.2 Control

This parameter shall indicate the Control attribute of the Program Invocation. This parameter may have the value CONTROLLING, CONTROLLED, or NORMAL. Depending on the value of this attribute, one of the following parameters shall be present.

7.3.3.9.2.3 Controlling

This parameter value shall be selected if the value of the Control attribute of the Program Invocation is CONTROLLING. If this parameter is selected, the following additional parameters shall appear.

7.3.3.9.2.4 List of Program Invocations

This parameter shall indicate the names of all the Program Invocations which are related to this Program Invocation by having their Reference to Controlling Program Invocation attribute value reference this Program Invocation. This list may have zero or more elements.

7.3.3.9.2.5 Program Location

This parameter, of type character string, shall indicate, if present, the location of the program execution if the Program Invocation is related to a sequential programming language. For those Program Invocations which are not related to a sequential programming language, this parameter shall be absent. Use of this parameter shall be described in the PICS (See 9.4).

7.3.3.9.2.6 Running Mode

This parameter shall indicate the value of the Running Mode attribute of the Program Invocation. Depending on the value of Running Mode one of the following parameters shall be present.

7.3.3.9.2.7 Free Running

Selection of this null parameter shall identify the Running Mode as FREE-RUN.

7.3.3.9.2.8 Remaining Cycle Count

Selection of this parameter shall identify the Running Mode as CYCLE-LIMITED. Further, the value of the parameter shall indicate the number of cycles remaining to be executed and is the value of the Remaining Cycle Count attribute of the Program Invocation.

7.3.3.9.2.9 Remaining Step Count

Selection of this parameter shall identify the Running Mode as STEP-LIMITED. Further, the value of the parameter shall indicate the number of steps remaining to be executed and shall be the value of the Remaining Step Count attribute of the Program Invocation. Implementation of the STEP-LIMITED Running Mode shall be defined in the PICS (See 9.4). If not supported, this choice may not be selected.

7.3.3.9.2.10 Controlled

This parameter value shall be selected if the value of the Control attribute of the Program Invocation is CONTROLLED. If this parameter is selected, the following additional parameter shall appear.

7.3.3.9.2.11 Controlling Program Invocation

This parameter shall be present if and only if the value of the Control attribute of the Program Invocation is CONTROLLED. If present, it shall indicate the name of the Program Invocation which is referenced by it Reference to Controlling Program Invocation attribute value. If no such Program Invocation is referenced, this parameter shall have the value UNCONTROLLED.

7.3.3.9.2.12 Normal

This parameter value shall be selected if the value of the Control attribute of the Program Invocation is NORMAL. If this parameter is selected, there are no additional parameters.

7.3.3.9.3 Extended service procedure

The service procedure is that specified in 11.9.2 of ISO/IEC 9506-1:1990. The robot specific information of the Attribute Detail shall be included in the response.

7.3.3.9.4 GetProgramInvocationAttributes protocol

The abstract syntax of the CS-GetProgramInvocationAttributes-Request and the CS-GetProgramInvocationAttributes-Response is specified below and described in the paragraphs that follow. Clause 5.5 of ISO/IEC 9506-2:1990 describes the derivation of all parameters for which explicit derivations are not provided in this clause.

```
CS-GetProgramInvocationAttributes-Request ::= NULL
```

```
CS-GetProgramInvocationAttributes-Response ::= SEQUENCE {
   errorCode     [0] IMPLICIT INTEGER,
   control       [1] CHOICE {
      controlling [0] IMPLICIT SEQUENCE {
         controlledPI    [0] IMPLICIT SEQUENCE OF Identifier,
         programLocation    [1] IMPLICIT VisibleString OPTIONAL,
         runningMode        [2] CHOICE {
               freeRunning     [0] IMPLICIT NULL,
               cycleLimited    [1] IMPLICIT INTEGER,
               stepLimited     [2] IMPLICIT INTEGER
                               }
                           },
      controlled  [1] CHOICE {
         controllingPI  [0] IMPLICIT Identifier, -- Reference to Controlling
         none           [1] IMPLICIT NULL -- uncontrolled
                        },
      normal       [2] IMPLICIT NULL
   }  }
```

7.3.3.9.4.1 Control

The value of the Control parameter shall be inferred from the selection made in the control element of the CS-GetProgramInvocationAttributes-Request. If the controlling choice is made, the value of the Control parameter shall be CONTROLLING. If the controlled choice is made, the value of the Control parameter shall be CONTROLLED. If the normal choice is made, the value of the Control parameter shall be NORMAL.

7.3.3.9.4.2 RunningMode

The value of the Running Mode parameter shall be inferred from the choice made under runningMode. The Running Mode parameter shall take on the value FREE-RUN if the freeRunning choice is selected in the service response. The Running Mode parameter shall take on the value of CYCLE LIMITED if the cycleLimited choice is selected in the service response. The Running Mode shall take on the value STEP LIMITED if the stepLimited choice is selected in the service response. For systems which do not support the STEP-LIMITED Running Mode, the runningMode choice shall be limited to values freeRunning ([0]), or cycleCount ([1]).

7.3.4 Other productions

The following productions, required for the MMS-General-Module (See A.1 of ISO/IEC 9506-2:1990), are not used in this part of ISO/IEC 9506, and are set equal to NULL. The services and protocol corresponding to these productions are fully defined in ISO/IEC 9506-1 and ISO/IEC 9506-2, and do not require robot specific extensions. They may be used directly in the abstract syntax defined in this part of ISO/IEC 9506.

```
CS-Input-Request  ::= NULL

CS-Output-Request ::= NULL

CS-InitiateDownloadSequence-Request ::= NULL
```

```
CS-DownloadSegment-Request ::= NULL

CS-TerminateDownloadSequence-Request ::= NULL

CS-InitiateUploadSequence-Request ::= NULL

CS-UploadSegment-Request ::= NULL

CS-TerminateUploadSequence-Request ::= NULL

CS-RequestDomainDownload-Request ::= NULL

CS-RequestDomainUpload-Request ::= NULL

CS-LoadDomainContent-Request ::= NULL

CS-StoreDomainContent-Request ::= NULL

CS-DeleteDomain-Request ::= NULL

CS-GetDomainAttributes-Request ::= NULL

CS-DefineEventCondition-Request ::= NULL

CS-DeleteEventCondition-Request ::= NULL

CS-GetEventConditionAttributes-Request ::= NULL

CS-ReportEventConditionStatus-Request ::= NULL

CS-AlterEventConditionMonitoring-Request ::= NULL

CS-TriggerEvent-Request ::= NULL

CS-DefineEventAction-Request ::= NULL

CS-DeleteEventAction-Request ::= NULL

CS-GetEventActionAttributes-Request ::= NULL

CS-ReportEventActionStatus-Request ::= NULL

CS-DefineEventEnrollment-Request ::= NULL

CS-DeleteEventEnrollment-Request ::= NULL

CS-AlterEventEnrollment-Request ::= NULL

CS-ReportEventEnrollmentStatus-Request ::= NULL

CS-GetEventEnrollmentAttributes-Request ::= NULL

CS-AcknowledgeEventNotification-Request ::= NULL

CS-GetAlarmSummary-Request ::= NULL

CS-GetAlarmEnrollmentSummary-Request ::= NULL

CS-ReadJournal-Request ::= NULL
```

```
CS-WriteJournal-Request ::= NULL

CS-InitializeJournal-Request ::= NULL

CS-ReportJournalStatus-Request ::= NULL

CS-CreateJournal-Request  ::= NULL

CS-DeleteJournal-Request ::= NULL

CS-GetCapabilityList-Request ::= NULL

CS-Input-Response ::= NULL

CS-Output-Response ::= NULL

CS-InitiateDownloadSequence-Response ::= NULL

CS-DownloadSegment-Response ::= NULL

CS-TerminateDownloadSequence-Response ::= NULL

CS-InitiateUploadSequence-Response ::= NULL

CS-UploadSegment-Response ::= NULL

CS-TerminateUploadSequence-Response ::= NULL

CS-RequestDomainDownload-Response ::= NULL

CS-RequestDomainUpload-Response ::= NULL

CS-LoadDomainContent-Response ::= NULL

CS-StoreDomainContent-Response ::= NULL

CS-DeleteDomain-Response ::= NULL

CS-GetDomainAttributes-Response ::= NULL

CS-DefineEventCondition-Response ::= NULL

CS-DeleteEventCondition-Response ::= NULL

CS-GetEventConditionAttributes-Response ::= NULL

CS-ReportEventConditionStatus-Response ::= NULL

CS-AlterEventConditionMonitoring-Response ::= NULL

CS-TriggerEvent-Response ::= NULL

CS-DefineEventAction-Response ::= NULL

CS-DeleteEventAction-Response ::= NULL

CS-GetEventActionAttributes-Response ::= NULL

CS-ReportEventActionStatus-Response ::= NULL
```

```
CS-DefineEventEnrollment-Response ::= NULL

CS-DeleteEventEnrollment-Response ::= NULL

CS-AlterEventEnrollment-Response ::= NULL

CS-ReportEventEnrollmentStatus-Response ::= NULL

CS-GetEventEnrollmentAttributes-Response ::= NULL

CS-AcknowledgeEventNotification-Response ::= NULL

CS-GetAlarmSummary-Response ::= NULL

CS-GetAlarmEnrollmentSummary-Response ::= NULL

CS-ReadJournal-Response ::= NULL

CS-WriteJournal-Response ::= NULL

CS-InitializeJournal-Response ::= NULL

CS-ReportJournalStatus-Response ::= NULL

CS-CreateJournal-Response ::= NULL

CS-DeleteJournal-Response ::= NULL

CS-GetCapabilityList-Response ::= NULL

CS-EventNotification ::= NULL

CsAdditionalObjectClasses ::= NULL

EN-Additional-Detail ::= NULL

EE-Additional-Detail ::= NULL

JOU-Additional-Detail ::= NULL

CS-GetNameList-Request ::= NULL

CS-GetNameList-Response ::= NULL

CS-Service-Error ::= NULL
```


7.4 Definition and use of robot specific services and protocol

7.4.1 Additional service requests and responses

7.4.1.1 General structure

ISO/IEC 9506-2 allows companion standards to define additional services for use solely within the companion standards. It does this by allowing for the specification of AdditionalService-Request within the ConfirmedServiceRequest production, the specification of AdditionalService-Response within the ConfirmedServiceResponse production, the specification of AdditionalUnconfirmedService within the UnconfirmedService production, and the specification of AdditionalService-Error within the ServiceError production.

This part of ISO/IEC 9506 defines four robot specific additional services. These services are the VMDStop, VMDReset, Select and AlterProgramInvocationAttributes services. These services are described in 7.4.2 through 7.4.5. The abstract syntax for these services is specified by the AdditionalService-Request and the AdditionalService-Response. There is no robot specific abstract syntax for the AdditionalUnconfirmedService or for the AdditionalService-Error.

7.4.1.2 Additional service requests

The abstract syntax for the AdditionalService-Request is defined below.

```
AdditionalService-Request ::= CHOICE {
    vMDStop     [0] IMPLICIT VMDStop-Request,
    vMDReset    [1] IMPLICIT VMDReset-Request,
    select      [2] IMPLICIT Select-Request,
    alterPI     [3] IMPLICIT AlterProgramInvocationAttributes-Request
    }
```

The AdditionalService-Request parameter shall identify the service type and the argument for that service. The context tags provided identify the service type. Definitions for each individual service specify the form of the argument for the service through definition of a type, which is referenced by the AdditionalService-Request production. Each of the services in the AdditionalService-Request choice is a confirmed service.

7.4.1.3 Additional service response

The abstract syntax for the AdditionalService-Response is defined below.

```
AdditionalService-Response ::= CHOICE {
    vMDStop     [0] IMPLICIT VMDStop-Response,
    vMDReset    [1] IMPLICIT VMDReset-Response,
    select      [2] IMPLICIT Select-Response,
    alterPI     [3] IMPLICIT AlterProgramInvocationAttributes-Response
    }
```

The AdditionalService-Response parameter shall identify the service type and the response for that service. The context tags provided identify the service type. Definitions for each individual service specify the form of the response for the service through definition of a type, which is referenced by the AdditionalService-Response production.

7.4.1.4 Additional service error

There are no additional service errors defined for this part of ISO/IEC 9506.

```
AdditionalService-Error ::= NULL
```

7.4.1.5 Additional unconfirmed service

There are no additional unconfirmed services defined for this part of ISO/IEC 9506.

```
AdditionalUnconfirmedService ::= NULL
```

7.4.2 VMDStop service

The VMDStop service provides for a way to stop all control activity at the responding robot and a way to put the robot into a state where manual intervention is required.

The VMDStop service is not intended to be equivalent to an Emergency Stop as defined in ISO DIS 10218. This service does not provide the same service as a hardwired emergency stop. Use of this service can result in similar actions from the robot, such as applying of brakes and removal of power to the motors. However, these actions are not required as part of this service.

7.4.2.1 Structure

The structure of the component service primitives is shown in Table 7.

Table 7 VMDStop service

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Result(+)			S	S(=)
Result(-) Error Type			S M	S(=) M(=)

7.4.2.1.1 Argument

There are no parameters in the argument of this service.

7.4.2.1.2 Result(+)

This parameter shall indicate that the requested service has succeeded. A successful result does not supply service specific parameters.

7.4.2.1.3 Result(-)

The Result(-) parameter shall indicate that the service request failed. The Error Type parameter, which is defined in detail in clause 17 of ISO/IEC 9506-1:1990 provides the reason for failure.

7.4.2.2 Service procedure

If the MMS user requesting this service does not own the R_CTRL semaphore, return a Result(-). Otherwise, the responding robot shall stop the robot motion and associated control activity as soon as possible and put the robot in the MANUAL-INTERVENTION-REQUIRED Robot VMD State. All Program Invocations in the RUNNING state shall be removed from the RUNNING state. The state of these Program Invocations is a local matter. The VMD logical status shall be set to NO-STATE-CHANGES-ALLOWED. These actions and their effect on the attributes of the VMD are summarized in Table 8.

Table 8 VMD attributes / VMDStop

VMD Attribute	State after Service Performed
Robot VMD State	MANUAL-INTERVENTION-REQUIRED
Local Control	undefined
VMD Logical State	NO-STATE-CHANGES-ALLOWED
VMD Physical State	NEEDS-COMMISSIONING

7.4.2.3 VMDStop protocol

The abstract syntax for the vMDStop choice of the AdditionalService-Request and AdditionalService-Response shall be specified by the VMDStop-Request and VMDStop-Response respectively. These types are specified below and described in the paragraphs that follow. Clause 5.5 of ISO/IEC 9506-2:1990 describes the derivation of all parameters for which explicit derivations are not provided in this clause.

```
VMDStop-Request ::= NULL

VMDStop-Response ::= NULL
```

7.4.3 VMDReset service

The VMDReset service provides a means to put the responding robot into an initialized state. This service also provides the ability for the client to request that the responding robot perform self-diagnostics in the initialization of the VMD. In this sense all information relating to physical status will have been validated as a result of this service.

In performing an initialization routine or in performing self-diagnostics the responding robot may not be able to maintain the application association. However, the association should be maintained, if possible. It is not the intent of this part of ISO/IEC 9506 for the connection to be broken as a result of this service.

7.4.3.1 Structure

The structure of the component service primitives is shown in Table 9.

Table 9 VMDReset service

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Extended Derivation	M	M(=)		
Result(+)			S	S(=)
Status Response			M	M(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.4.3.1.1 Argument

This parameter contains the parameter of the VMDReset service request.

7.4.3.1.1.1 Extended Derivation

This parameter, of type boolean, indicates whether (true) or not (false) to perform self-diagnostics as part of the VMDReset service.

7.4.3.1.2 Result(+)

This parameter shall indicate that the requested service has succeeded. The Result(+) parameter for this service includes the Status Response parameter used in the MMS Status service. The robot specific attributes of the VMD shall be included in this response.

7.4.3.1.3 Result(-)

The Result(-) parameter shall indicate that the service request failed. The Error Type parameter, which is defined in detail in clause 17 of ISO/IEC 9506-1:1990 provides the reason for failure.

7.4.3.2 Service procedure

Upon receipt of a VMDReset service indication, the responding robot shall perform the following steps:

a) If the R_CTRL semaphore is not owned by the MMS user requesting this service, return a Result(-).

b) If the robot is in the ROBOT-EXECUTING state, return a Result(-).

c) If local conditions prevent completion of this service, return a Result(-).

d) Delete all Program Invocations and Domains which are MMS Deletable.

e) Place the robot into the ROBOT-IDLE state.

f) Perform a Status Service procedure (See 9.2.2 of ISO/IEC 9506-1:1990) using the Extended Derivation parameter.

If any step of this procedure fails, the service fails and a Result(-) response shall be returned. Otherwise a Result(+) shall be returned containing the Status Response as augmented by the Robot Status Detail which is described in 7.3.2.1.1.

7.4.3.3 VMDReset protocol

The abstract syntax for the vMDReset choice of the AdditionalService-Request and AdditionalService-Response is specified by the VMDReset-Request and VMDReset-Response respectively. These types are specified below and described in the paragraphs that follow. Clause 5.5 of ISO/IEC 9506-2:1990 describes the derivation of all parameters for which explicit derivations are not provided in this clause.

```
VMDReset-Request ::= BOOLEAN -- Extended Derivation

VMDReset-Response ::= Status-Response
```

7.4.4 Select service

The Select service provides a means to identify a Program Invocation as the Selected Program Invocation controlling the Robot VMD. It also binds the Controlling Program Invocation to a set of Controlled Program Invocations. The Select service also allows the deselection of a Controlling Program Invocation and the removal of all Controlled Program Invocations from the List of References to Controlled Program Invocations.

7.4.4.1 Structure

The structure of the component service primitives is shown in Table 10.

Table 10 Select service

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Controlling Program Invocation	U	U(=)		
List of Program Invocations	C	C(=)		
Result(+)			S	S(=)
Result(-)			S	S(=)
Error Type			M	M(=)

7.4.4.1.1 Argument

This parameter contains the parameters of the Select service request.

7.4.4.1.1.1 Controlling Program Invocation

This parameter, of type identifier, indicates the Controlling Program Invocation which is to be selected. The Control attribute of this Program Invocation shall have a value equal to CONTROLLING. If this parameter is not present, it indicates that the present Controlling Program Invocation is to be deselected, and no Program Invocation is to be selected.

7.4.4.1.1.2 List of Program Invocations

This parameter, of type list of identifier, indicates the Program Invocations which will be placed under the control of the referenced Controlling Program Invocation. This parameter shall be present only if the Controlling Program Invocation parameter is present.

7.4.4.1.2 Result(+)

This parameter shall indicate that the requested service has succeeded. A successful result does not supply service specific parameters.

7.4.4.1.3 Result(-)

The Result(-) parameter shall indicate that the service request failed. The Error Type parameter, which is defined in detail in clause 17 of ISO/IEC 9506-1:1990 provides the reason for failure.

7.4.4.2 Service procedure

Upon receipt of a Select service request, the responding MMS user shall perform the following steps:

a) If the R_CTRL semaphore is not owned by the MMS user requesting this service, return a Result(-).

b) If the Controlling Program Invocation parameter is present and the Program Invocation indicated by the Controlling Program Invocation parameter does not have its Control attribute equal to CONTROLLING, return a Result(-) and skip the remainder of this procedure.

c) If the Controlling Program Invocation parameter is present and the Program Invocation indicated by the Controlling Program Invocation parameter is not in the IDLE state, return a Result(-) and skip the remainder of this procedure.

d) For each element of the List of References to Controlled Program Invocations, verify that the Program Invocation has its control attribute equal to CONTROLLED and that the value of the Reference to Controlling Program Invocation attribute is equal to UNCONTROLLED or equal to the Reference to Selected Program Invocation attribute of the VMD. If any element of the list does not meet this test, return a Result(-) and skip the remainder of this procedure.

e) If the Reference to Selected Program Invocation attribute of the VMD is not NONE, verify that the Program Invocation is in the IDLE state. If this is not true, return a Result(-) and skip the remainder of this procedure.

f) If the Reference to Selected Program Invocation attribute of the VMD is not NONE, perform the following steps for the Program Invocation referenced by this attribute.

   1) For each Controlled Program Invocation on the List of References to Controlled Program Invocations of the Controlling Program Invocation, set the Reference to Controlling Program Invocation to UNCONTROLLED.

   2) Set the List of References to Controlled Program Invocations to an empty list.

g) Set the Reference to Selected Program Invocation attribute of the VMD to NONE.

h) If the Controlling Program Invocation parameter is not present, return Result(+) and skip the remainder of this procedure.

i) If the List of References to Controlled Program Invocations attribute of the Program Invocation indicated by the Controlling Program Invocation parameter is not empty, return Result(-) and skip the remainder of this procedure.

j) Set the Reference to Selected Program Invocation attribute of the VMD equal to a reference to the Program Invocation indicated by the Controlling Program Invocation parameter.

k) If the List of Program Invocations parameter is empty, set the List of References to Controlled Program Invocations attribute to an empty list.

l) If the List of Program Invocations parameter is not empty, then for each element of the list, set the Reference to Controlling Program Invocation attribute to reference the Program Invocation indicated by the Controlling Program Invocation parameter. Add this Program Invocation to the List of References to Controlled Program Invocation attribute of the Controlling Program Invocation.

m) Return a Result(+).

7.4.4.3 Select protocol

The abstract syntax for the select choice of the AdditionalService-Request and AdditionalService-Response is specified by the Select-Request and Select-Response respectively. These types are specified below and described in the paragraphs that follow. Clause 5.5 of ISO/IEC 9506-2:1990 describes the derivation of all parameters for which explicit derivations are not provided in this clause.

```
Select-Request ::= SEQUENCE {
   controlling [0] IMPLICIT Identifier OPTIONAL,
   controlled  [1] IMPLICIT SEQUENCE OF Identifier OPTIONAL
   -- this field shall appear if and only if the controlling field is included
   }

Select-Response ::= NULL
```

7.4.5 AlterProgramInvocationAttributes service

The AlterProgramInvocationAttributes service provides a means to alter certain attribute values of a Program Invocation.

7.4.5.1 Structure

The structure of the component service primitives is shown in Table 11.

Table 11 AlterProgramInvocationAttributes service

Parameter Name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Controlling Program Invocation	M	M(=)		
Running Mode	C	C(=)		
No Limit	S	S(=)		
Cycle Count	S	S(=)		
Step Count	S	S(=)		
			S	S(=)
Result(+)				
			S	S(=)
Result(-)			M	M(=)
Error Type				

7.4.5.1.1 Argument

This parameter contains the parameters of the AlterProgramInvocationAttributes service request.

7.4.5.1.1.1 Controlling Program Invocation

This parameter, of type Identifier, shall identify the Program Invocation whose attributes are to be altered. This Program Invocation shall have its Control attribute equal to CONTROLLING.

7.4.5.1.1.2 Running Mode

The value of this parameter shall indicate the value for the Running Mode attribute of the Program Invocation. Depending on which Running Mode of the Program Invocation is selected, one of the following parameters shall be present.

7.4.5.1.1.3 No Limit

This parameter, of type null, shall be chosen if the value of the Running Mode attribute is to be set to FREE-RUN.

7.4.5.1.1.4 Cycle Count

This parameter, of type integer, shall be chosen if the value of the Running Mode attribute is to be set to CYCLE-LIMITED, and the value of the Remaining Cycle Count attribute shall be set to this parameter value.

7.4.5.1.1.5 Step Count

This parameter, of type integer, shall be chosen if the value of the Running Mode attribute is to be set to STEP-LIMITED, and the value of the Remaining Step Count attribute shall be set to this parameter value. Implementation of the STEP-LIMITED Running Mode shall be defined in the PICS (See 9.4). If not supported, this choice may not be selected.

7.4.5.2 Service procedure

The following service procedure shall apply to the AlterProgramInvocationAttributes service.

a) If the requesting MMS user does not own the R_CTRL semaphore, a Result(-) shall be returned.

b) If the Control attribute of the Program Invocation is not equal to CONTROLLING, return a Result(-) and skip the remainder of this procedure.

c) The attributes of the Controlling Program Invocation shall be set as follows:

   1) If the No Limit parameter is selected in the service request, the Running Mode attribute of the Program Invocation shall be set to FREE-RUN.

   2) If the Cycle Count parameter is selected in the service request the Running Mode attribute of the Program Invocation shall be set to CYCLE-LIMITED and the Remaining Cycle Count attribute of the Program Invocation shall be set to the value of the Cycle Count parameter.

   3) If the Step Count parameter is selected in the service request the Running Mode attribute of the Program Invocation shall be set to STEP-LIMITED and the Remaining Step Count attribute of the Program Invocation shall be set to the value of the Step Count parameter.

7.4.5.3 AlterProgramInvocationAttributes protocol

The abstract syntax of the AlterProgramInvocationAttributes-Request and AlterProgramInvocationAttributes-Response is specified below and described in the paragraphs that follow. Clause 5.5 of ISO/IEC 9506-2:1990 describes the derivation of all parameters for which explicit derivations are not provided in this clause.

```
AlterProgramInvocationAttributes-Request ::= SEQUENCE {
   programInvocation [0] IMPLICIT Identifier,
   startCount        [1] IMPLICIT StartCount DEFAULT cycleCount 1 }

AlterProgramInvocationAttributes-Response ::= NULL
```

7.4.5.3.1 Running Mode

The value of the Running Mode parameter shall be inferred by the choice made within the StartCount type. The choice of noLimit shall indicate that the Running Mode shall be set to FREE-RUN and the No Limit parameter shall be selected, the choice of cycleCount shall indicate that the Running Mode shall be set to CYCLE-LIMITED and the Cycle Count parameter shall be selected, and the choice of stepCount shall indicate that the Running Mode shall be set to STEP-LIMITED and the Step Count parameter shall be selected.

The abstract syntax of StartCount shall be inferred from the selection made in the value of the Running Mode parameter such that:

a) the value of StartCount shall be the noLimit choice if No Limit is selected.

b) the value of StartCount shall be the cycleCount choice if Cycle Count is selected.

c) the value of StartCount shall be the stepCount choice if Step Count is selected.

7.5 The Initiate service and protocol

7.5.1 Init Request Detail parameter

ISO/IEC 9506-2 provides for a Init Request Detail parameter to be defined by companion standards. The structure of the Init Request Detail parameter is shown in Table 12.

Table 12 Init Request Detail

Parameter Name	Req	Ind
Proposed Version Number	M	M(=)
Proposed Parameter CBB	M	M(=)
Services Supported Calling	M	M(=)
Additional Services Supported Calling	M	M(=)

This parameter shall be present if the abstract syntax defined in this part of ISO/IEC 9506 is proposed for this instance of communication. Otherwise, it shall not be present. It shall contain parameters relating to communication in the presentation context derived from the abstract syntax defined in this part of ISO/IEC 9506. The component parameters are specified as follows:

7.5.1.1 Proposed Version Number

This parameter, of type integer, shall contain a number which represents a minor version number of this part of ISO/IEC 9506. This number is the proposed minor version number which will be used in the presentation context derived from the abstract syntax defined in this part of ISO/IEC 9506 for this instance of communication. Proposal of a number greater than one indicates support for all minor versions between one and the number proposed.

Major revisions of this part of ISO/IEC 9506 are reflected through the definition and registration of distinct abstract syntaxes. (See clause 17 of ISO/IEC 9506-2:1990) Minor revisions are reflected in the minor version number parameter. Minor versions of this part of ISO/IEC 9506 at the same major revision level are compatible with versions of this part of ISO/IEC 9506 with smaller minor version numbers.

The value of this parameter may be reduced by the MMS provider if it can not support the requested value. The value in the indication primitive shall be less than or equal to the value in the request primitive, but not less than one.

7.5.1.2 Proposed Parameter CBB

This parameter is specified in 8.2.3.2 of ISO/IEC 9506-1:1990.

7.5.1.3 Services Supported Calling

This parameter is specified in 8.2.3.3 of ISO/IEC 9506-1:1990.

7.5.1.4 Additional Services Supported Calling

This parameter, of type bitstring, shall specify support by the Calling MMS user of a set of additional services which are defined by this part of ISO/IEC 9506 for use in the presentation context derived from the abstract syntax defined in this part of ISO/IEC 9506 on the application association.

The value of the parameter in the indication primitive shall specify the intersection of the set of additional services supported by the Calling MMS user and the set of services supported by the MMS provider.

The assignment of a service to an individual bit of the bitstring type is specified in 7.5.3. A value of one in the assigned bit shall indicate support for the corresponding service. A value of zero shall indicate non-support. Any additional bits shall be ignored.

Support for confirmed services shall be defined as the ability to receive a request indication and properly execute the service procedure defined for the responder role.

If a confirmed service is supported, then a Reject PDU shall not be issued on receipt of that service, except for a protocol error. If a confirmed service is not supported, then a Reject PDU shall be issued on receipt of that service with a reject code of "UNRECOGNIZED SERVICE".

7.5.2 Init Response Detail parameter

ISO/IEC 9506-2 provides for a Init Response Detail parameter to be defined by companion standards. The structure of the InitResponse Detail parameter is shown in Table 13.

Table 13 Init Response detail

Parameter Name	Req	Ind
Negotiated Version Number	M	M(=)
Negotiated Parameter CBB	M	M(=)
Services Supported Called	M	M(=)
Additional Services Supported Called	M	M(=)

This parameter shall be present if the abstract syntax defined in this part of ISO/IEC 9506 has been negotiated for this instance of communication. Otherwise, it shall not be present. It shall contain parameters relating to communication in the presentation context derived from the abstract syntax defined in this part of ISO/IEC 9506. The component parameters are specified as follows:

7.5.2.1 Negotiated Version Number

This parameter, of type integer, shall contain a number which represents a minor version number of this part of ISO/IEC 9506. This number is the minor version number of this part of ISO/IEC 9506 which will be used in the presentation context derived from -the abstract syntax defined in this part of ISO/IEC 9506 for this instance of communication. This number shall be less than or equal to the Proposed Version Number parameter in the request primitive. It shall not be reduced to less than one.

Major revisions of this part of ISO/IEC 9506 are reflected through the definition and registration of distinct abstract syntaxes. (See clause 17 of ISO/IEC 9506-2:1990) Minor revisions are reflected in the minor version number parameter. Minor versions of this part of ISO/IEC 9506 at the same major revision level are compatible with versions of this part of ISO/IEC 9506 with smaller minor version numbers.

7.5.2.2 Negotiated Parameter CBB

This parameter is specified in 8.2.4.2 of ISO/IEC 9506-1:1990.

7.5.2.3 Services Supported Called

This parameter is specified in 8.2.4.3 of ISO/IEC 9506-1:1990.

7.5.2.4 Additional Services Supported Called

This parameter, of type bitstring, shall specify support by the Called MMS user of a set of additional services which are defined in this part of ISO/IEC 9506 for use in the presentation context derived from the abstract syntax defined in this part of ISO/IEC 9506 on the application association.

The value of the parameter in the indication primitive shall specify the intersection of the set of additional services supported by the Called MMS user and the set of additional services supported by the MMS provider.

The assignment of a service to an individual bit of the bitstring type is specified in 7.5.3. A value of one in the assigned bit shall indicate support for the corresponding service. A value of zero shall indicate non-support. Any additional bits shall be ignored.

Support for confirmed services shall be defined as the ability to receive a request indication and properly execute the service procedure defined for the responder role.

If a confirmed service is supported, then a Reject PDU shall not be issued on receipt of that service, except for a protocol error. If a confirmed service is not supported, then a Reject PDU shall be issued on receipt of that service with a reject code of "UNRECOGNIZED SERVICE".

7.5.3 Initiate protocol

The abstract syntax of the Init Request Detail and Init Response Detail parameters shall be specified by the InitRequestDetail and InitResponseDetail types respectively. These types are specified below and described in the paragraphs that follow. Clause 5.5 of ISO/IEC 9506-2:1990 describes the derivation of all parameters for which explicit derivations are not provided in this subclause.

```
InitRequestDetail ::= SEQUENCE {
    proposedVersionNumber       [0] IMPLICIT Integer16,
    proposedParameterCBB        [1] IMPLICIT ParameterSupportOptions,
    servicesSupportedCalling    [2] IMPLICIT ServiceSupportOptions,
    additionalSupportedCalling  [3] IMPLICIT AdditionalSupportOptions
    }

InitResponseDetail ::= SEQUENCE {
    negotiatedVersionNumber     [0] IMPLICIT Integer16,
    negotiatedParameterCBB      [1] IMPLICIT ParameterSupportOptions,
    servicesSupportedCalled     [2] IMPLICIT ServiceSupportOptions,
    additionalSupportedCalled   [3] IMPLICIT AdditionalSupportOptions
    }
```

```
AdditionalSupportOptions ::= BITSTRING {
   vMDStop                          (0),
   vMDReset                         (1),
   select                           (2),
   alterProgramInvocationAttributes (3)
   }
```

7.6 End of module

The following END statement closes the module.

```
END
```

8 Robot specific standardized objects

8.1 General

This clause defines names and other attributes of standardized MMS visible entities that are specific to robots and generic to all classes of robots. The standardized objects refer to several different kinds of entities including Types, Domains, Program Invocations, Named Variables, Semaphores, Event Conditions and Event Actions.

Each standardized object defined in this clause has a standardized meaning associated with it. Not all the objects listed are considered appropriate to be supported by all robot devices. However, if a object listed in this clause is supported in a robot, then it shall be supported with the semantics specified in this part of ISO/IEC 9506. Conversely, if functionality which is identical with the semantics associated with one of these standardized objects occur in an implementation, the standardized name shall be used to identify that functionality. It is expected that non-standard names will also be defined by the implementor or user in the course of developing applications. Use of these non-standard names is a local matter.

All standardized names defined in this clause are prefixed with "R_" (R Underscore), the R indicating a member of this part of ISO/IEC 9506, and the underscore indicating a standardized name.

Note: This clause includes definitions of Named Type objects. While such objects are not normally visible in an implementation, they are included in order (1) to provide a convenient notation for complex types used by more than one standardized Named Variable object and (2) to provide guidance to implementors on the preferred structure for local Named Variable objects.

8.2 Standardized Domain objects

8.2.1 R_ARM

This name shall be used to identify the Domain associated with the robot arm. The Domain models the Robot Arm object described in 6.2. For robotic systems with more than one robot arm, each arm shall be represented by a separate Domain, named R_ARM_1, R_ARM_2, and so on.

Object: Domain
Key Attribute: Domain Name = "R_ARM"
Attribute: List of Capabilities
Attribute: State
Attribute: MMS Deletable = FALSE
Attribute: Sharable = TRUE
Attribute: Domain Content
  Attribute: List of Subordinate Objects
Attribute: List of Program Invocation References = {R_ARM}

8.2.2 R_CAL

This name shall be used to identify the Domain associated with the calibration procedure. The calibration operation is described in 6.3.6.

Object: Domain
Key Attribute: Domain Name = "R_CAL"
Attribute: List of Capabilities
Attribute: State
Attribute: MMS Deletable = FALSE
Attribute: Sharable = FALSE
Attribute: Domain Content
  Attribute: List of Subordinate Objects
Attribute: List of Program Invocation References = {R_CAL}

8.2.3 R_SAFE

This name shall be used to identify the Domain associated with safety related equipment including, but not limited to, Emergency Stop switches, light gates, pressure sensitive mats, etc.

Object: Domain
Key Attribute: Domain Name = "R_SAFE"
Attribute: List of Capabilities
Attribute: State
Attribute: MMS Deletable = FALSE
Attribute: Sharable = FALSE
Attribute: Domain Content
  Attribute: List of Subordinate Objects
Attribute: List of Program Invocation References

8.3 Standardized Program Invocation objects

8.3.1 R_ARM

This name identifies the Program Invocation which manipulates the robot arm. This Program Invocation is permanently associated with the R_ARM Domain.

Object: Program Invocation
Key Attribute: Program Invocation Name = "R_ARM"
Attribute: State
Attribute: List of Domain References = {R_ARM}
Attribute: MMS Deletable = FALSE
Attribute: Reusable = TRUE
Attribute: Monitor Attribute: Execution Argument
Attribute: Error Code = 0
Attribute: Control = CONTROLLED
  Attribute: Reference to Controlling Program Invocation

8.3.2 R_CAL

This name identifies the Program Invocation which performs the calibration procedure. This Program Invocation is permanently associated with the R_CAL Domain. The calibration procedure is described in 6.3.6.

Object: Program Invocation
Key Attribute: Program Invocation Name = "R_CAL"
Attribute: State
Attribute: List of Domain References = {R_CAL}
Attribute: MMS Deletable = FALSE
Attribute: Reusable = TRUE
Attribute: Monitor
Attribute: Execution Argument
Attribute: Error Code = 0
Attribute: Control = NORMAL

8.4 Standardized Named Type objects

8.4.1 R_PIS - position in space

A data value which represents a position in space shall be described in terms of this type which is a three element array of floating point values. A position is described in terms of X, Y and Z coordinates as described in 5.2.2. The array represents the coordinates of the X, Y, and Z dimensions of the point. The X, Y, and Z coordinate axis are orthogonal to each other and their positive direction complies with the right-hand rule.

```
Object: Named Type
Key Attribute: Type Name = vmd-specific "R_PIS"
Attribute: MMS Deletable = FALSE
Attribute: Type Description = array
   { numberOfElements 3,
     floating-point { format-width 32, exponent-width 8 } }
```

8.4.2 R_OIS - orientation in space

A data value which represents an orientation in space shall be described in terms of this type. R_OIS is a four element array of floating point values. The meaning of these values shall be provided in the PICS (See 9.4).

```
Object: Named Type
Key Attribute: Type Name = vmd-specific "R_OIS"
Attribute: MMS Deletable = FALSE
Attribute: Type Description = array
   { numberOfElements 4,
     floating-point { format-width 32, exponent-width 8 } }
```

Note: The intent is to accommodate Euler angles, direction cosines, or quaternions. When an international standard is established on the representation of the orientation in space, this type will be altered to represent that definition.

8.4.3 R_PSE - pose

A data value which represents both a position and an orientation in space shall be described in terms of this type. This type is a concatenation of R_PIS and R_OIS.

```
Object: Named Type
Key Attribute: Type Name = vmd-specific "R_PSE"
Attribute: MMS Deletable = FALSE
Attribute: Type Description = structure
  { componentType typeName vmd-specific "R_PIS",
    componentType typeName vmd-specific "R_OIS" }
```

The relationship between joint coordinates and robot pose is, in general, not one to one. In order to make the relationship unambiguous, additional information is needed. Presently, there is no standard which represents this information, although work on this subject is in progress. When such a standard has been achieved, it will be appropriate to provide a standardized type for describing this information.

8.4.4 R_EEF - end effector

A data value which represents the attributes of an end effector shall be described in terms of this type. This type can be used to describe the end effector which is part of the robot arm (See 8.5.5.15) or of other end effectors which are part of the system.

```
Object: Named Type
Key Attribute: Type Name = vmd-specific "R_EEF"
Attribute: MMS Deletable = FALSE
Attribute: Type Description = structure { components {
  { componentName "IDNumber",
    componentType objId },
  { componentName "ToolDescriptor",
    componentType visible-string 80},
  { componentName "TOOL-MICS",
    componentType typeName vmd-specific "R_PSE"},
  { componentName "User-Detail",
    componentType visible-string 512} } }
```

8.5 Standardized Named Variable objects

8.5.1 General definitions

The model of a robot system developed in clause 5, distinguishes between

a) data values which are transferred between the task program and the robot arm control program referred to as Programmed values,

b) data values which are the output of the path planner in the robot arm referred to as Commanded values,

c) the data values which corresponding to the real position of the robot joints referred to as Actual values.

Within this part of ISO/IEC 9506, the following naming convention is used: If the object represents a Programmed value, the variable name is of the form "R_Pxxx". If the object represents a Commanded value, the variable name is of the form "R_Cxxx". If the object represents an Actual value, the variable name is of the form "R_Axxx".

If the object represents the Pose (position and orientation) of some object with respect to some coordinate system, the variable name is of the form "R_Txx", indicating that the variable corresponds to a transformation of one coordinate system to another. The convention further indicate which coordinate systems are transformed to which;

the letters T, U, B, M and W refer to Tool Coordinate System, User Coordinate System, Base Coordinate System, Mechanical Interface Coordinate System, or World Coordinate System. Thus R_TMB refers to the position of the Mechanical Interface with respect to the Base Coordinate System, or equivalently, to the transformation which will carry the Base Coordinate System into the Mechanical Interface Coordinate System.

8.5.2 Generic Named Variable objects

All generic Named Variable objects listed in clause 18 of ISO/IEC 9506-1:1990 shall be supported if the associated function is supported.

8.5.3 VMD specific standardized Named Variable objects

8.5.3.1 R_VPWR - power

This Named Variable object represents the value of the Any Physical Resource Power On attribute of the VMD. This boolean variable shall be TRUE if power is applied to any physical resource in the VMD, FALSE otherwise.

Object: Named Variable
Key Attribute: Variable Name = vmd-specific "R_VPWR"
Attribute: MMS Deletable = FALSE
Attribute: Type Description = boolean
Attribute: Access Method

8.5.3.2 R_VCAL - calibrated

This Named Variable object represents the value of the All Physical Resources Calibrated attribute of the VMD.

Object: Named Variable
Key Attribute: Variable Name = vmd-specific "R_VCAL"
Attribute: MMS Deletable = FALSE
Attribute: Type Description = boolean
Attribute: Access Method

8.5.3.3 R_VLOCAL - VMD Local Control

This Named Variable object represents the value of the Local Control attribute of the VMD.

Object: Named Variable
Key Attribute: Variable Name = vmd-specific "R_VLOCAL"
Attribute: MMS Deletable = FALSE
Attribute: Type Description = boolean
Attribute: Access Method

8.5.3.4 R_VSAFEV - Safety Interlocks Violated

This Named Variable object represents the state of the Safety Interlocks Violated attribute of the VMD.

Object: Named Variable
Key Attribute: Variable Name = vmd-specific "R_VSAFEV"
Attribute: MMS Deletable = FALSE
Attribute: Type Description = boolean
Attribute: Access Method

8.5.4 Domain-specific standardized Named Variable objects

8.5.4.1 R_DPWR - Device Power On

This Named Variable object represents the value of the DevicePowerOn attribute of the Robot Arm or other Auxiliary Device object. This object has the value of TRUE if power is applied to the device.

Object: Named Variable
Key Attribute: Variable Name = domain-specific {
domainID ,
itemID "R_DPWR" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = boolean
Attribute: Access Method

8.5.4.2 R_DCAL - Device Calibrated

This Named Variable object represents the value of the DeviceCalibrated attribute of the Robot Arm object or other Auxiliary Device object. This object has the value of zero if the device resource is CALIBRATED, one if the device resource is NOTCALIBRATED, and two if the device resource is CALIBRATING.

Object: Named Variable
Key Attribute: Variable Name = domain-specific {
domainID ,
itemID "R_DCAL" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = integer 8
Attribute: Access Method

8.5.4.3 R_DLOCAL - Device Local Control

This Named Variable object represents the value of the Local Control attribute of the Robot Arm object or other Auxiliary Device object. This named object is TRUE if the device resource is in local control, FALSE otherwise.

Object: Named Variable
Key Attribute: Variable Name = domain-specific {
domainID ,
itemID "R_DLOCAL" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = boolean
Attribute: Access Method

8.5.5 R_ARM Domain-specific standardized Named Variable objects

8.5.5.1 R_NJ - Number of Joints

This Named Variable object represents the value of the Number of Joints attribute of the Robot Arm object. This value is an integer which specifies the number of joints associated with a specific robot arm.

Object: Named Variable
Key Attribute: Variable Name = domain-specific {
domainID "R_ARM",
itemID "R_NJ" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = integer 8
Attribute: Access Method

8.5.5.2 R_JT - Joint Types

This Named Variable object represents the value of the Joint Type attribute of the Robot Arm object. This object is an array of integers which identify the types of the joints of the robot arm. An array element of zero indicates a revolute joint, an element of one indicates a prismatic joint. The array size is the number of joints in the robot, R_NJ. The ordering of the joint information corresponding to each joint in the robot is a local matter.

```
Object: Named Variable
Key Attribute: Variable Name = domain-specific {
                    domainID  "R_ARM",
                    itemID    "R_JT" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = array
    { numberOfElements r-NJ-value, elementType integer 8 }
Attribute: Access Method
```

8.5.5.3 R_JCAL - Joints Calibrated

This Named Variable object represents the value of the Calibrated attribute for each of the joints in the Ordered List Of Joint Descriptions of the Robot Arm object. This array contains the value of the Calibrated attribute for each of the joints. The ordering of the joint information corresponding to each joint in the robot is a local matter.

```
Object: Named Variable
Key Attribute: Variable Name = domain-specific {
                    domainID "R_ARM",
                    itemID   "R_JCAL" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = array
    { numberOfElements r-NJ-value, elementType boolean }
Attribute: Access Method
```

8.5.5.4 R_JBK - Brakes On

This Named Variable object represents the value of the Joint Brakes attribute of the Robot Arm object. This array contains the value of the brakes status for each of the joints. An array element of zero indicates that no brakes exist for the corresponding joint (the Joint Brakes attribute is FALSE), an element of one indicates that the value of the Brakes On attribute is FALSE and an element of two indicates that the value of the Brakes On attribute is TRUE. The ordering of the joint information corresponding to each joint in the robot is a local matter.

```
Object: Named Variable
Key Attribute: Variable Name = domain-specific {
                    domainID "R_ARM",
                    itemID   "R_JBK" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = array
    { numberOfElements r-NJ-value, elementType integer 8 }
Attribute: Access Method
```

8.5.5.5 R_JBD - Joint Bounds

This Named Variable object represents the value of the Upper Bound and Lower Bound attribute of the Robot Arm object. This array contains the upper and lower bounds on each of the joint coordinates. The ordering of the joint information corresponding to each joint in the robot is a local matter.

```
Object: Named Variable
Key Attribute: Variable Name = domain-specific {
                    domainID "R_ARM",
                    itemID   "R_JBD" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = array
    { numberOfElements r_NJ-value, elementType structure
   { components {
   { componentName "upperBound",
    componentType floating-point
    { format-width 32, exponent-width 8 }
    }
   { componentName "lowerBound",
    componentType floating-point
    { format-width 32, exponent-width 8 }
    } } } }
Attribute: Access Method
```

8.5.5.6 R_AJV - Actual Joint Values

This Named Variable object represents the value of the Actual Joint Value attribute of the Robot Arm object. This array contains the actual values of the joint coordinates. The ordering of joint information corresponding to each joint in the robot is a local matter.

```
Object: Named Variable
Key Attribute: Variable Name = domain-specific {
                    domainID "R_ARM",
                    itemID   "R_AJV" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = array
  { numberOfElements r-NJ-value,
    elementType floating-point
    { format-width 32, exponent-width 8 } }
Attribute: Access Method
```

8.5.5.7 R_TUB - coordinate transformation - User to base

This Named Variable object represents the value of the USER-BASE attribute of the Robot Arm object. This value is the pose of the User expressed in the Base Coordinate System, or equivalently the transformation of the Base Coordinate System of the robot arm to that of the User. It is of type Pose.

Object: Named Variable
Key Attribute: Variable Name = domain-specific {
domainID "R_ARM",
itemID "R_TUB" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = typeName vmd-specific "R_PSE"
Attribute: Access Method

8.5.5.8 R_TTM - coordinate transformation - Tool to MICS

This Named Variable object represents the value of the TOOL-MICS attribute of the Robot Arm object. This value is the pose of the Tool expressed in the Mechanical Interface Coordinate System, or equivalently, the transformation of the Mechanical Interface Coordinate System to that of the Tool. It is of type Pose.

Object: Named Variable
Key Attribute: Variable Name = domain-specific {
domainID "R_ARM",
itemID "R_TTM" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = typeName vmd-specific "R_PSE"
Attribute: Access Method

8.5.5.9 R_TBW - coordinate transformation - Base to World

This Named Variable object represents the value of the BASE-WORLD attribute of the Robot Arm object. This value is the pose of the Base Coordinate System expressed in the World Coordinate System, or equivalently, the transformation of the World Coordinate System to that of the Base. It is of type Pose.

Object: Named Variable
Key Attribute: Variable Name = domain-specific {
domainID "R_ARM",
itemID "R_TBW" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = typeName vmd-specific "R_PSE"
Attribute: Access Method

8.5.5.10 R_ATUP - actual Tool User point

This Named Variable object represents the value of the actual pose of the Tool in the User Coordinate System. It is derived from the Robot Arm attributes, TOOL-MICS, MICS-BASE, USER-BASE, Actual Joint Value and implementation specific parameters.

Object: Named Variable
Key Attribute: Variable Name = domain-specific {
domainID "R_ARM",
itemID "R_ATUP" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = typeName vmd-specific "R_PSE"
Attribute: Access Method

8.5.5.11 R_CTUP - commanded Tool User point

This Named Variable object represents the value of the commanded TOOL-USER attribute of the Robot Arm object. This object contains the value of the destination pose of the Tool in the User Coordinate System. This is either the actual pose if no motion is underway, or the pose to which the robot arm has been commanded most recently.

Object: Named Variable
Key Attribute: Variable Name = domain-specific {
domainID "R_ARM",
itemID "R_CTUP" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = typeName vmd-specific "R_PSE"
Attribute: Access Method

8.5.5.12 R_AMBP - actual Mechanical Interface to Base position

This Named Variable object represents the value of the MICS-BASE attribute of the Robot Arm object, which represents the actual position of the MICS with respect to the base of the robot. It is derived from the Robot Arm attributes, Actual Joint Value and implementation specific parameters.

Object: Named Variable
Key Attribute: Variable Name = domain-specific {
domainID "R_ARM",
itemID "R_AMBP" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = typeName vmd-specific "R_PSE"
Attribute: Access Method

8.5.5.13 R_SF - Speed Factor

This Named Variable object represents the value of the Speed Factor attribute of the Robot Arm object. This object contains the value of the ratio of the programmed tool tip speed to the commanded speed. It is a dimension less number, normally unity.

Object: Named Variable
Key Attribute: Variable Name = domain-specific {
domainID "R_ARM",
itemID "R_SF" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = floating-point
{ format-width 32, exponent-width 8 }
Attribute: Access Method

8.5.5.14 R_PSTU - Programmed Speed of Tool - User coordinate system

This Named Variable object represents the value of the Programmed Speed attribute of the Robot Arm object. This object contains the programmed value of the tool speed.

Object: Named Variable
Key Attribute: Variable Name = domain-specific {
domainID "R_ARM",
itemID "R_PSTU" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = floating-point
{ format-width 32, exponent-width 8 }
Attribute: Access Method

8.5.5.15 R_EEFU - End Effector in use

This Named Variable object represents the value of the attributes of the End Effector currently in use. It is itself an attribute of the Robot Arm object.

Object: Named Variable
Key Attribute: Variable Name = domain-specific {
domainID "R_ARM",
itemID "R_EEFU" }
Attribute: MMS Deletable = FALSE
Attribute: Type Description = typeName vmd-specific "R_EEF"
Attribute: Access Method

8.6 Standardized Semaphore objects

8.6.1 R_CTRL

This VMD-Specific Semaphore object may be used to represent a semaphore which has control over the robot arm resource(s). Its applicability is in multiple host systems in which exclusive control, by one host, of all of the physical resources of the VMD is required.

Object: Semaphore
Key Attribute: Semaphore Name = vmd-specific "R_CTRL"
Attribute: MMS Deletable = FALSE
Attribute: Class = TOKEN
  Attribute: Number Of Tokens = 1
  Attribute: Number Of Owned Tokens
Attribute: List Of Owners
Attribute: List Of Requesters

8.7 Standardized Event Condition objects

8.7.1 R_RVS Robot VMD state change

This Event Condition object identifies the Event Condition of the VMD attribute Robot VMD State changing from EXECUTING to any other value.

Object: Event Condition
Key Attribute: Event Condition Name = vmd-specific "R_RVS"
Attribute:MMS Deletable = FALSE
Attribute: Event Condition Class = MONITORED
Attribute: State = (initially ACTIVE)
Attribute: Priority = normalPriority
Attribute: Severity = normalSeverity
Attribute: Additional Detail = NULL
Attribute: List of Event Enrollment References
  Attribute: Enabled = TRUE
  Attribute: Alarm Summary Reports = FALSE
  Attribute: Monitored Variable Reference = UNSPECIFIED
  Attribute: Evaluation Interval = (local matter)
  Attribute: Time of Last Transition to Active
  Attribute: Time of Last Transition to Idle

8.7.2 R_SIV robot Safety Interlocks Violated

This Event Condition object identifies the Event Condition when the VMD attribute Safety Interlocks Violated transitions from FALSE to TRUE.

Object: Event Condition
Key Attribute: Event Condition Name = vmd-specific "R_SIV"
Attribute:MMS Deletable = FALSE
Attribute: Event Condition Class = MONITORED
Attribute: State = ACTIVE
Attribute: Priority = normalPriority
Attribute: Severity = normalSeverity
Attribute: Additional Detail = NULL
Attribute: List of Event Enrollment References
  Attribute: Enabled = TRUE
  Attribute: Alarm Summary Reports = FALSE
  Attribute: Monitored Variable Reference = UNSPECIFIED
  Attribute: Evaluation Interval = (local matter)
  Attribute: Time of Last Transition to Active
  Attribute: Time of Last Transition to Idle

8.7.3 R_RLC robot Local Control changed

This Event Condition object identifies the Event Condition when the VMD attribute Local Control changes value.

Object: Event Condition
Key Attribute: Event Condition Name = vmd-specific "R_RLC"
Attribute: MMS Deletable = FALSE
Attribute: Event Condition Class = MONITORED
Attribute: State = ACTIVE
Attribute: Priority = normalPriority
Attribute: Severity = normalSeverity
Attribute: Additional Detail = NULL
Attribute: List of Event Enrollment References
  Attribute: Enabled = TRUE
  Attribute: Alarm Summary Reports = FALSE
  Attribute: Monitored Variable Reference = UNSPECIFIED
  Attribute: Evaluation Interval = (local matter)
  Attribute: Time of Last Transition to Active
  Attribute: Time of Last Transition to Idle

8.7.4 R_ARM - Robot arm operation

This Event Condition object identifies the Event Condition when the R_ARM Program Invocation leaves the RUNNING state.

Object: Event Condition
Key Attribute: Event Condition Name = vmd-specific "R_ARM"
Attribute: MMS Deletable = FALSE
Attribute: Event Condition Class = MONITORED
Attribute: State = ACTIVE
Attribute: Priority = normalPriority
Attribute: Severity = normalSeverity
Attribute: Additional Detail = NULL
Attribute: List of Event Enrollment References
  Attribute: Enabled = TRUE
  Attribute: Alarm Summary Reports = FALSE
  Attribute: Monitored Variable Reference = UNSPECIFIED
  Attribute: Evaluation Interval = (local matter)
  Attribute: Time of Last Transition to Active
  Attribute: Time of Last Transition to Idle

8.8 Standardized Event Action objects

8.8.1 R_STC robot Status change

This Event Action object identifies the action taken when one of the named events occurs. This action will provide Event Notification with the MMS Status service result.

Object: Event Action
Key Attribute: Event Action Name = vmd-specific "R_STC"
Attribute: MMS Deletable = FALSE
Attribute: Confirmed Service Request =
  Status { Extended Derivation = FALSE }
Attribute: List of Modifier = NULL
Attribute: List of Event Enrollment References

8.8.2 R_ARM - Robot arm operation

This Event Action object identifies the action taken when the R_ARM Program Invocation leaves the RUNNING state. This action will provide Event Notification with the MMS GetProgramInvocationAttributes service result.

Object: Event Action
Key Attribute: Event Action Name = vmd-specific "R_ARM"
Attribute: MMS Deletable = FALSE
Attribute: Confirmed Service Request =
  GetProgramInvocationAttributes { "R_ARM" }
Attribute: List of Modifier = NULL
Attribute: List of Event Enrollment References

8.9 Standardized Event Enrollment objects

This part of ISO/IEC 9506 does not define any Standardized Event Enrollment objects for robot specific use.

8.10 Standardized Operator Station objects

This part of ISO/IEC 9506 does not define any Standardized Operator Station objects for robot specific use.

8.11 Standardized Journal objects

This part of ISO/IEC 9506 does not define any Standardized Journal objects for robot specific use.

9 Robot conformance classes

9.1 General

This clause lists a set of MMS conformance classes for robot applications. The conformance classes are based on models of robot applications. The list of applications are indicated by combining the class name with the type of program loading method. Upload/download is for systems which can support remote uploading and downloading of their devices directly. Secondary storage is for systems which can only be loaded from their local file store. All classes contain the Base Conformance Class. ISO/IEC 9506-1 describes conformance on the basis of the behaviour of the server or VMD.

Flexible application of robots requires that they be connected to a network and have their activities coordinated with other equipment. As described earlier, there are two fundamental modes of operation of a robot - as a client device and as a server device. The robot may function in either or both modes simultaneously. However, the services and conformance requirements of the robot when acting as a client and responding as a server, are considered separately.

This clause describes the minimal set of services and conformance requirements for a robot to function as a server device. Also presented are the services and conformance requirements for enhanced performance of robots acting as client and server devices.

9.2 Robot server conformance

9.2.1 General requirements

Minimum requirements for all MMS responders are prescribed in 17.9 of ISO/IEC 9506-2:1990. These minimum requirements includes support for the responder role for

Initiate,
Conclude,
Abort,
Reject and
Identify.

This requirement applies with equal force to any implementation claiming conformance to the abstract syntax defined in this part of ISO/IEC 9506.

In the conformance classes which follow, the robot is acting as an MMS server connected to a remote host computer which acts as the MMS client.

9.2.2 Base Conformance Class

9.2.2.1 Static conformance requirements

The Base Conformance Class specifies the minimum set of services that a robot shall provide to conform to this part of ISO/IEC 9506. This conformance class forms a kernel of services that are included in all other robot conformance classes.

This conformance class is for robot servers that support only one association at a time. Robot servers that support more than one association are considered to be in a higher conformance class (i.e. Multiple Hosts Class). These higher conformance classes have additional requirements.

For the purposes of satisfying the requirements of the Robot Model described in clause 5, of the MMS model mapping described in clause 6, and of the elements of the service procedures described in clause 7, the remote user proposing or accepting establishment of an application association specifying the abstract syntax defined in this part of ISO/IEC 9506 shall have control of the R_CTRL semaphore immediately upon successful establishment of the association. No explicit MMS service procedures for semaphores need to be supported in order to satisfy this conformance class. This automatic allocation of the R_CTRL semaphore to the remote MMS user does not apply to any higher conformance classes in which multiple associations may be supported.

9.2.2.2 Service conformance requirements

9.2.2.2.1 Fixed requirements

The following services shall be supported in the Base Conformance Class.

Cancel
Status
UnsolictedStatus
GetDomainAttributes
DeleteDomain
CreateProgramInvocation
DeleteProgramInvocation
Start
Stop
Resume
Reset
GetProgramInvocationAttributes
Read
Write
InformationReport
Select

In addition to these services, a robot shall support either the set of services specified in 9.2.2.2.2.1 or in 9.2.2.2.2.2.

9.2.2.2.2 Loading of Domains

9.2.2.2.2.1 Upload/Download

This set of services shall be supported by robots that use remote loading of their robot system controller from a host system. These services provide a way to upload and download programs and data to/from the robot system controller.

InitiateDownloadSequence
DownloadSegment
TerminateDownloadSequence
InitiateUploadSequence
UploadSegment
TerminateUploadSequence
RequestDomainDownload
RequestDomainUpload