INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.744.1
(03/2003)

SERIES X: DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

OSI management – Management functions and ODMA functions

CORBA-based TMN software management service

ITU-T Recommendation X.744.1

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

PUBLIC DATA NETWORKS	
Services and facilities	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalling and switching	X.50–X.89
Network aspects	X.90–X.149
Maintenance	X.150–X.179
Administrative arrangements	X.180–X.199
OPEN SYSTEMS INTERCONNECTION	
Model and notation	X.200–X.209
Service definitions	X.210–X.219
Connection-mode protocol specifications	X.220–X.229
Connectionless-mode protocol specifications	X.230–X.239
PICS proformas	X.240–X.259
Protocol Identification	X.260–X.269
Security Protocols	X.270–X.279
Layer Managed Objects	X.280–X.289
Conformance testing	X.290–X.299
INTERWORKING BETWEEN NETWORKS	
General	X.300–X.349
Satellite data transmission systems	X.350–X.369
IP-based networks	X.370–X.399
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	X.500–X.599
OSI NETWORKING AND SYSTEM ASPECTS	
Networking	X.600–X.629
Efficiency	X.630–X.639
Quality of service	X.640–X.649
Naming, Addressing and Registration	X.650–X.679
Abstract Syntax Notation One (ASN.1)	X.680–X.699
OSI MANAGEMENT	
Systems Management framework and architecture	X.700–X.709
Management Communication Service and Protocol	X.710–X.719
Structure of Management Information	X.720–X.729
Management functions and ODMA functions	**X.730–X.799**
SECURITY	X.800–X.849
OSI APPLICATIONS	
Commitment, Concurrency and Recovery	X.850–X.859
Transaction processing	X.860–X.879
Remote operations	X.880–X.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999

For further details, please refer to the list of ITU-T Recommendations.

ITU-T Recommendation X.744.1

CORBA-based TMN software management service

Summary

This Recommendation is part of a series of Recommendations that specify the CORBA interface requirements for communication between an Operations System (OS) and a Network Element (NE), between an OS and a Mediation Device (MD), between an OS and a Q Adapter (QA), and between OSs in a Telecommunication Management Network (TMN). This Recommendation proposes a CORBA-based Software Management implementation based on ITU-T Rec. X.744. The intent of this Recommendation is to define a CORBA/IDL model similar to that defined in ITU-T Rec. X.744 using CMISE. In this Recommendation, both fine-grained and coarse-grained (i.e., facade) interfaces are defined for the software management functions. This Recommendation is compliant with CORBA modelling standards from ITU-T Recs X.780, X.780.1, Q.816, Q.816.1 and M.3120.

Source

ITU-T Recommendation X.744.1 was prepared by ITU-T Study Group 4 (2001-2004) and approved under the WTSA Resolution 1 procedure on 29 March 2003.

CONTENTS

Page

1 Scope 1

2 References........ 2

3 Definitions 4

4 Abbreviations........ 4

5 Requirements for Software Management 5

6 Model for Software Management function........ 5

6.1 Software Management capabilities........ 5

6.2 Relationships among managed objects........ 7

6.3 Software Unit managed object 9

6.4 Executable Software managed object........ 16

6.5 Software Distributor managed object........ 17

6.6 Use of Universal Identifiers (UIDs) 19

6.7 Relationships 20

7 Relationship with other functions........ 20

8 Compliance and conformance 20

8.1 System conformance 21

8.2 Conformance statement guidelines........ 21

9 ITU-T Rec. X.744.1 IDL listing........ 22

9.1 Imports........ 22

9.2 Forward declarations 23

9.3 Structures and typedefs........ 23

9.4 Exceptions 35

9.5 Software unit 36

9.6 SoftwareUnit interface........ 38

9.7 SoftwareUnit_F interface 47

9.8 SoftwareUnitFactory interface 56

9.9 Executable software........ 57

9.10 ExecutableSoftware interface........ 57

9.11 ExecutableSoftware_F interface........ 58

9.12 ExecutableSoftwareFactory interface........ 59

9.13 Software distributor........ 60

9.14 SoftwareDistributor interface 60

9.15 SoftwareDistributor_F interface........ 62

9.16 SoftwareDistributorFactory interface........ 63

9.17 Notifications 64

9.18 Name binding 66

9.19 FileTypeConst module 69

9.20 DeliverResultConst module........ 70

ITU-T Recommendation X.744.1

CORBA-based TMN software management service

1 Scope

This Recommendation is part of a series of Recommendations that specify the CORBA interface requirements for communication between an Operations System (OS) and a Network Element (NE), between an OS and a Mediation Device (MD), between an OS and a Q Adapter (QA), and between OSs in a Telecommunication Management Network (TMN) [1].

The Software Management Function includes management of a system for delivery of software and also management of software within a system.

There are two aspects of software that need to be considered separately. These two aspects can be described as the "dormant" view and the "active" view of software.

The dormant view of software is related to the data that is stored in a managed system and the way in which it is delivered and installed. In general, the data is stored information, such as data files and tables, but may also be files containing executable code. The scope of this Recommendation includes the dormant view of software.

The active view of software is related to the management of resources that utilize the software. There is no real difference between this view and the normal view of management of resources. The scope of this Recommendation does not include the active view of software. However, the relationship between managed objects representing resources that utilize software, and the managed objects representing the software that they are using (i.e., dormant view of software) is within the scope of this Recommendation.

The scope of this Recommendation includes:

– initiation of transfer of software;
– post transfer control of software;
– software activation (includes version update and patching);
– software de-activation;
– software reversion change;
– software validation;
– software enquiry;
– software backup;
– software restore.

The scope of this Recommendation does not include:

– transfer mechanism of software;
– physical storage of software (mapping of software to physical file store, such as to a floppy disk, hard disk, etc.);
– formatting of software;
– naming of software products;
– sequencing of software management commands;
– software monitoring;
– management of processes running in a system.

In this Recommendation, both fine-grained and coarse-grained (i.e., facade) interfaces are defined for the software management functions. The fine-grained and facade interfaces are providing the same support for software management. Both can be used with fine-grained and coarse-grained managed objects (e.g., those defined in ITU-T Rec. Rec. M.3120 [2]).

Current telecommunications networks are populated by a large and increasing number of OSs and network elements supplied by different vendors. Both the number and variety of networks and services have grown, creating a diversity of management needs. This growth has resulted in the proliferation of unique communication interfaces between OSs and network elements. The telecommunications industry stands to benefit from the standardization of these interfaces, designed to achieve interoperability between a broad range of OSs and network elements/Q Adapters, using Mediation Devices where appropriate, and between OSs.

The primary purpose of this Recommendation is to provide a set of application messages and associated support objects for the support of communication across CORBA interfaces. Because of the desirability of providing common TMN solutions, these messages and support objects are expected to be applicable to other TMN or TMN-related interfaces.

2 References

The following ITU-T Recommendations and other references contain provisions, which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is published regularly. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[1] ITU-T Recommendation M.3010 (2000), *Principles for a telecommunications management network*.

[2] ITU-T Recommendation M.3120 (2001), *CORBA generic network and network element level information model*.

[3] ITU-T Recommendation Q.816 (2001), *CORBA-based TMN services*.

[4] ITU-T Recommendation Q.816 (2001), *CORBA-based TMN services* plus Amendment 1 (2001): *OMG services profile*.

[5] ITU-T Recommendation Q.816 (2001), *CORBA-based TMN services* plus Amendment 2 (2002): *User guide for local name resolution*.

[6] ITU-T Recommendation Q.816 (2001), *CORBA-based TMN services* plus Corrigendum 1 (2001).

[7] ITU-T Recommendation Q.816.1 (2001), *CORBA-based TMN services*: *Extensions to support coarse-grained interfaces*.

[8] ITU-T Recommendation Q.822.1 (2001), *CORBA-based TMN performance management service*.

[9] ITU-T Recommendation X.701 (1997), *Information technology – Open Systems Interconnection – Systems management overview*.

[10] ITU-T Recommendation X.720 (1992), *Information technology – Open Systems Interconnection – Structure of management information: Management information model*.

[11] ITU-T Recommendation X.721 (1992), *Information technology – Open Systems Interconnection – Structure of management information: Definition of management information*.

[12] ITU-T Recommendation X.722 (1992), *Information technology – Open Systems Interconnection – Structure of management informations: Guidelines for the definition of managed objects*.

[13] ITU-T Recommendation X.723 (1993), *Information technology – Open Systems Interconnection – Structure of management information: Generic management information*.

[14] ITU-T Recommendation X.731 (1992), *Information technology – Open Systems Interconnection – Systems management: State management function*.

[15] ITU-T Recommendation X.740 (1992), *Information technology – Open Systems Interconnection – Systems management: Security audit trail function*.

[16] ITU-T Recommendation X.741 (1995), *Information technology – Open Systems Interconnection – Systems management: Objects and attributes for access control*.

[17] ITU-T Recommendation X.742 (1995), *Information technology – Open Systems Interconnection – Systems management: Usage metering function for accounting purposes*.

[18] ITU-T Recommendation X.744 (1996), *Information technology – Open Systems Interconnection – Systems management: Software management function*.

[19] ITU-T Recommendation X.744 (1996), *Information technology – Open Systems Interconnection – Systems management: Software management function* plus Technical Corrigendum 1 (1998).

[20] ITU-T Recommendation X.744 (1996), *Information technology – Open Systems Interconnection – Systems management: Software management function* plus Technical Corrigendum 2 (2000).

[21] ITU-T Recommendation X.744 (1996), *Information technology – Open Systems Interconnection – Systems management: Software management function* plus Technical Corrigendum 3 (2001).

[22] ITU-T Recommendation X.745 (1993), *Information technology – Open Systems Interconnection – Systems management: Test management function*.

[23] ITU-T Recommendation X.746 (2000), *Information technology – Open Systems Interconnection – Systems management: Scheduling function*.

[24] ITU-T Recommendation X.780 (2001), *TMN guidelines for defining CORBA managed objects*.

[25] ITU-T Recommendation X.780 (2001), *TMN guidelines for defining CORBA managed objects* plus Corrigendum 1 (2001).

[26] ITU-T Recommendation X.780 (2001), *TMN guidelines for defining CORBA managed objects* plus Corrigendum 2 (2002).

[27] ITU-T Recommendation X.780 (2001), *TMN guidelines for defining CORBA managed objects* plus Amendment 1 (2002)*, System objects and user guide for bulk attribute retrieval*.

[28] ITU-T Recommendation X.780.1 (2001), TMN *guidelines for defining coarse-grained CORBA managed object interfaces*.

[29] ITU-T Recommendation X.780.1 (2001), *TMN guidelines for defining coarse-grained CORBA managed object interfaces* plus Amendment 1 (2002)*, System façades and user guide for bulk attribute retrieval*.

[30] ITU-T Recommendation X.780.1 (2001), *TMN guidelines for defining coarse-grained CORBA managed object interfaces* plus Corrigendum 1 (2002).

3 Definitions

This Recommendation uses the following terms from other Recommendations:

3.1 This Recommendation uses the following terms defined in ITU-T Rec. M.3010 [1]:

a) Management Information Model

b) Manager

3.2 This Recommendation uses the following term defined in ITU-T Rec. X.701 [9]:

a) Managed Object Class

3.3 This Recommendation uses the following terms defined in ITU-T Rec. X.744 [18]:

a) Backup

b) Deliver

c) Execute

d) Install

e) Restore

f) Revert

g) Utilize

h) Validate

3.4 This Recommendation uses the following terms defined in ITU-T Rec. X.780 [24]:

a) Inheritance Hierarchy

b) Naming Tree

c) Subordinate Objects

d) Superior Object

4 Abbreviations

This Recommendation uses the following abbreviations:

CORBA	Common Object Request Broker Architecture
IDL	Interface Definition Language
MD	Mediation Device
NE	Network Element
OMG	Object Management Group
ORB	Object Request Broker
OS	Operations System
QA	Q Adapter
TMN	Telecommunications Management Network
UID	Universal Identifier

5 Requirements for Software Management

Software Management must be able to satisfy the following requirements, subject to possible imposed controls and conditions:

a) be able to request delivery of software to a specific managed system;

b) be able to control the installation of software on a managed system, including the installation of patches (e.g., upgrades) and to revert back to a previous version of the software;

c) be able to initiate the execution of a software program;

d) be able to inquire as to the attributes of all software held on a managed system;

e) be able to create and delete software held on a managed system;

f) be able to validate software held on a managed system in order to check its integrity;

g) be able to restrict use of software resources on a managed system for administrative purposes;

h) be able to back up a software item and to restore a previously backed up software item.

The model shall not preclude the use of logging, accounting, auditing and license management at each state of this process.

In all cases the success or failure of the operation needs to be reported to the managing system.

Relationships exist between software managed objects that reflect the way in which software resources utilize other software resources. This relationship is known as "utilization," and is used to indicate which versions of software are to be used at any time. The management of this relationship is for further study.

6 Model for Software Management function

The Software Management Service is primarily concerned with managing pieces of software. In this context software includes data, control information and executable instructions. The management view of software may be represented by a managed object of class Software Unit. The management of software representing executable instructions has additional characteristics and is represented by the Executable Software managed object (which is a sub-class of Software Unit).

Another aspect of software management is the management of the delivery of software to the managed system. Software is not necessarily managed in the same units as it is delivered. For example, it is possible that a number of different software units may well be delivered together to a managed system (e.g., on a CD-ROM). Similarly, if a large software unit is to be delivered over a communications network, it may well be necessary to break it up into smaller parts for the purpose of delivery. Another case concerns the delivery of only the changes required to an existing software object, for example a patch. Management of delivery is done in terms of the Software Distributor managed object.

6.1 Software Management capabilities

6.1.1 Create

A Software Unit may be created on a managed system using the Create operation. A Software Unit may be created either in the Created or the Delivered state, depending on behaviour. A Software Unit may also be created as a result of the Delivery operation on a Software Distributor managed object or as a result of some local action. As a result of being created, a Software Unit managed object may emit an Object Creation notification.

6.1.2 Delete

A Software Unit may be deleted from a managed system using the Delete operation. A side effect of the Delete operation on a Software Unit may be the removal of associated underlying resources. As a result of being deleted, a Software Unit managed object may emit an Object Deletion notification.

6.1.3 Deliver

Delivery of a coordinated set of Software Units may be requested. The result of delivery indicates the success or failure. Delivery is accomplished by sending the Deliver operation to the Software Distributor object. The result of a Deliver operation is that a copy of the target software items is delivered to the target system in the Delivered state. A side effect may be the creation of one or more associated objects (e.g., Software Units).

Packaging of the software units and the choice of transfer mechanism is a local matter and outside the scope of this Recommendation. For example, this information may be pre-configured or specified in the Deliver operation along with any other associated information.

6.1.4 Execute Program

A managing system may request the execution of Executable Software via the Execute Program operation. This Recommendation does not specify how such an execution of the Executable Software may be subsequently managed, but merely provides a mechanism for initiating the execution of the Executable Software.

6.1.5 Get

It is possible to find out information about software (what software is present, what software is available for use, relationships between software, etc.) using the Get operation. If successful, the result of Get operation contains the information requested.

6.1.6 Install

Installation customizes the software for use. This can potentially require a significant amount of processing and time, and may involve checking that all parts of the target software version are present on the managed system (whether they have been delivered as part of an update or are already present) and assembling them ready for use. Installation is accomplished via the Install operation directed to the Software Unit managed object, where it is an optional characteristic.

A patch is a modification to software and may be represented by a Software Unit managed object. Therefore, a patch may be delivered, installed, utilized, copied, etc., using software management.

A special case of installation is that of a "patch," where it is the patch that is delivered, and the application of the patch is the installation and produces an updated version of the software ready for utilization. The updated version receives an updated version number for identification, delivered with the patch. The result of an installation indicates its success or failure.

When delivery of the software is complete, installation may commence. The installation may need to be coordinated; however, such coordination is outside the scope of this Recommendation. Examples of coordination include:

– completing delivery of several software items before another software item may be installed;
– checking that the current software is not in use before enabling a new one;
– checking that particular versions of other software are (not) installed;
– synchronizing installation with other open systems, providing for simultaneous but uncoupled installation on more than one open system, providing a way for installation on more than one open system to be bound together in a single unit of work, etc.

6.1.7 Revert

The Revert operation is used to cause an installed Software Unit managed object to revert back to being not installed or to cause one or more applied patches to be removed. The Revert operation is accomplished via a method directed to the Software Unit managed object, where it is an optional characteristic.

It may be necessary to maintain additional information regarding how a Software Unit managed object may be reverted. However, this is outside the scope of this Recommendation.

6.1.8 Set Administrative State

Once software has been customized for use (installed), the software may be made available for use via the Set Administrative State to Unlocked operation directed to the Software Unit managed objects. It is possible to remove specified software from a state of availability by setting the Administrative State of the Software Unit managed object to Shutting Down or Locked.

6.1.9 Validation

The integrity of software may be checked using the Validate operation. It is also possible to validate that software previously delivered remains in a usable condition. The result of validation indicates whether the software was validated. The Validate operation is accomplished via an operation directed to the Software Unit managed object, where it is an optional characteristic.

6.1.10 Notifications

All the operations applicable to a Software Unit may require that results be sent to the managing system along with the confirmation that the operation has been completed. It may also be necessary to notify another managing system of completion of one of these operations when the request for the operation was not originated from that managing system.

6.1.11 Backup

The Backup operation may be used by a managing system to request performing a backup on a target object. The Backup operation when applied to software managed object causes a copy of the underlying resources to be made. It has no direct effect on the original software resources.

6.1.12 Restore

The Restore operation may be used by a managing system to request performing a restore of a previously backed-up target object.

6.2 Relationships among managed objects

Figure 1 contains the inheritance hierarchy for the Software Management managed objects and Figure 2 contains the Software Management naming tree hierarchy. Note that the System managed objects may have other name bindings defined than to a Managed Element managed object.

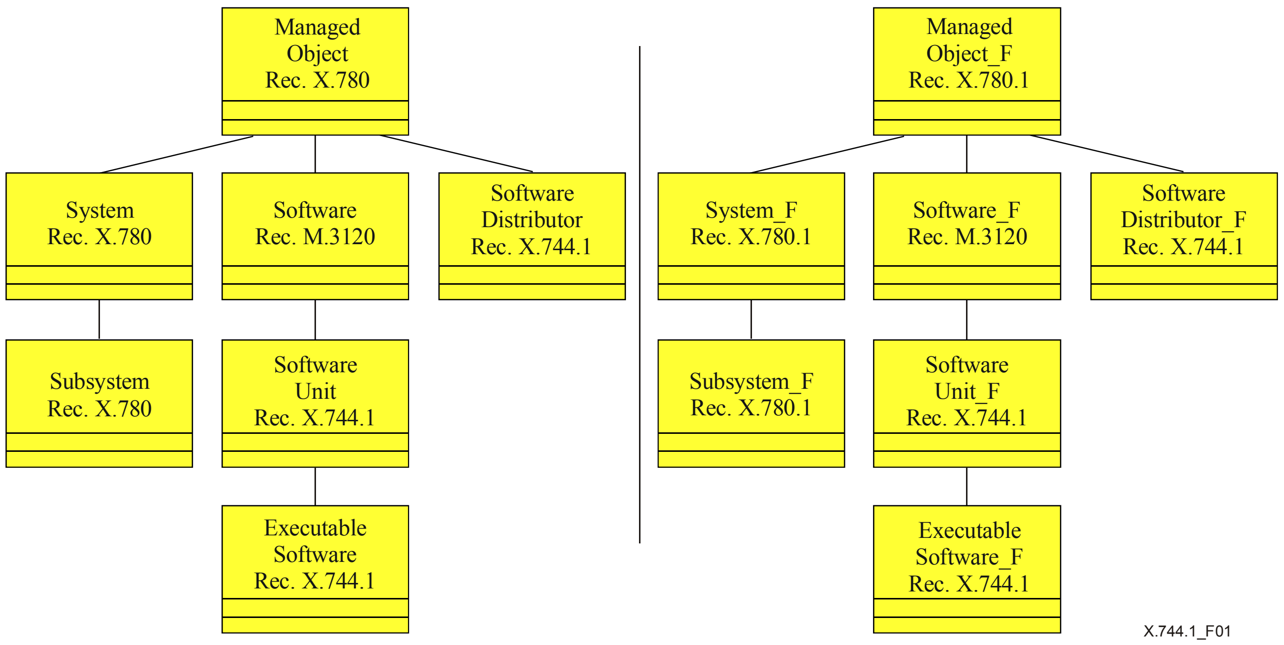
Managed Object Rec. X.780
System Rec. X.780
Software Rec. M.3120
Software Distributor Rec. X.744.1
Subsystem Rec. X.780
Software Unit Rec. X.744.1
Executable Software Rec. X.744.1
Managed Object_F Rec. X.780.1
System_F Rec. X.780.1
Software_F Rec. M.3120
Software Distributor_F Rec. X.744.1
Subsystem_F Rec. X.780.1
Software Unit_F Rec. X.744.1
Executable Software_F Rec. X.744.1
X.744.1_F01

Figure 1/X.744.1 – Software Management inheritance hierarchy

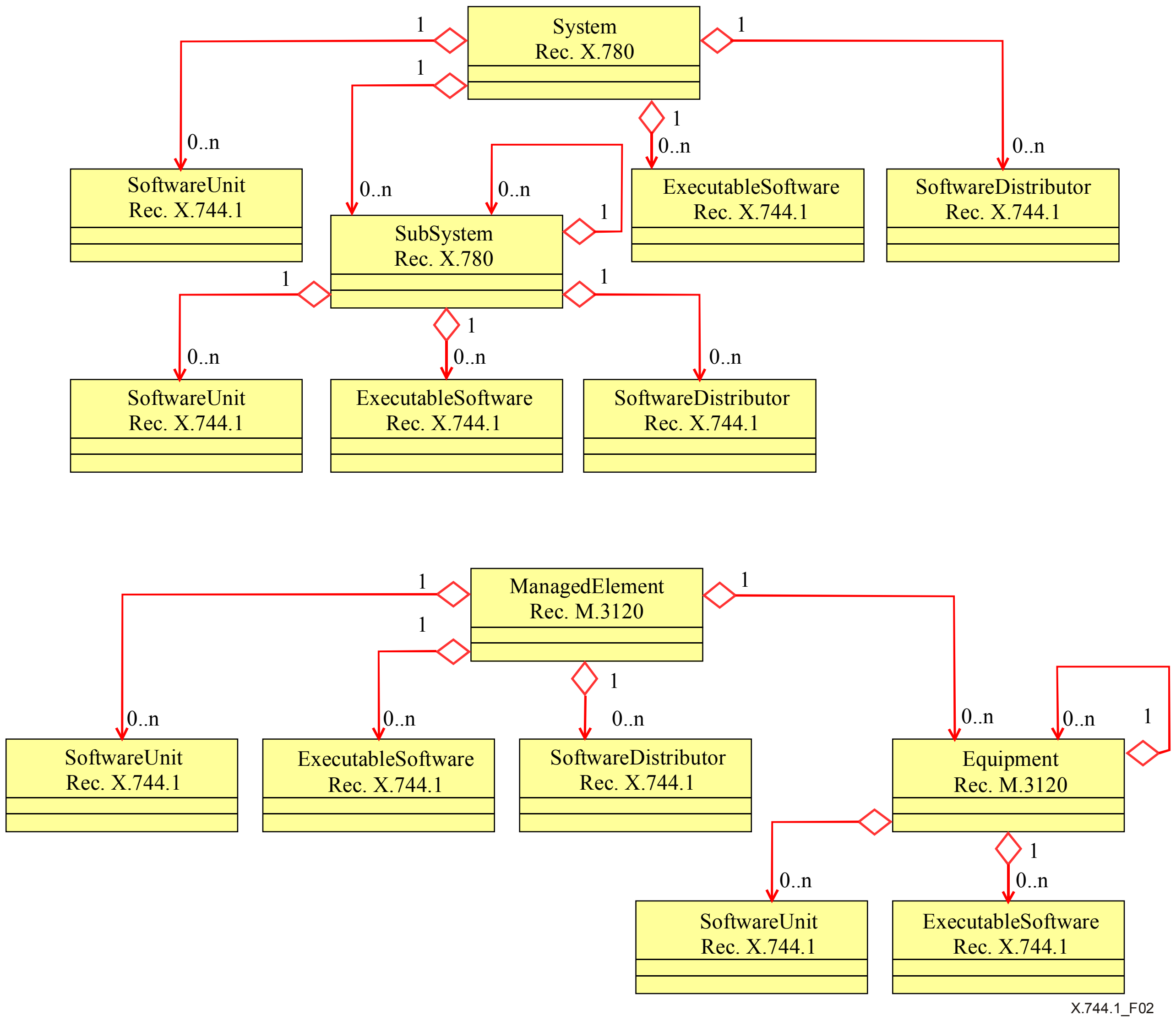
System Rec. X.780
SoftwareUnit Rec. X.744.1
SubSystem Rec. X.780
ExecutableSoftware Rec. X.744.1
SoftwareDistributor Rec. X.744.1
ManagedElement Rec. M.3120
Equipment Rec. M.3120
1
0..n
X.744.1_F02

Figure 2/X.744.1 – Software Management naming tree hierarchies

6.3 Software Unit managed object

6.3.1 Creating Software Unit managed objects

When a Software Unit managed object is first created it represents the resource that is prepared to accept delivery of a piece of software.

There are three mechanisms or operations by which a Software Unit managed object may be created:

1) Software may be replicated onto a target managed system without any operation (e.g., by local means).
2) Use of the Create operation in the Software Unit Factory to create a Software Unit managed object in the Created or Delivered state.
3) Use of the Deliver operation directed to the Software Distributor managed object in the source system. This may create Software Unit managed objects in the target destination in the Delivered state. The Deliver operation applied to a Software Distributor managed object causes the software to be delivered to a target destination. When all the required software has been successfully delivered, a Deliver Result notification is emitted by the Software Distributor managed object to signify that delivery has been completed.

The Create operation causes the creation of a Software Unit managed object on the target destination. Success of creation would be indicated by the existence of a Software Unit managed object on the target destination (an Object Creation notification may also be emitted).

If previously created but not delivered, the Software Unit managed object may emit an Attribute Value Change notification when delivery is completed, indicating that its internal state has changed from "Created" to "Delivered". A Software Unit managed object may also be instantiated in the Delivered state.

Once the software is in the Delivered state, the next stage is for the software to become installed. Installation involves customizing the software for use. For example, installation may involve the preparation and customizing of the software unit for an upgrade, perhaps by taking a copy of the current software resources and applying the changes to the copy. Installation may involve setting up configuration and dependency relationships between the Software Unit and other software objects.

Once software has been installed, it may be utilized by other managed objects. It may need to be made available for use by setting the Administrative State to Unlocked if it is in the Locked state. This operation is implemented by using the Administrative State Set operation on the Administrative State attribute.

Making the software unavailable for use is accomplished by administratively locking it (i.e., set Administrative State to Locked or Shutting Down). This prevents any new processes from using the software. If the Administrative State is set to Locked, all currently running processes using this software are aborted. If the Administrative State is set to Shutting Down, the currently running process is allowed to keep running but new processes are not allowed. When all the currently running processes using this software have completed execution, the Administrative State becomes Locked. This operation is implemented by using the Administrative State Set operation on the Administrative State attribute.

The Revert operation may be used to reverse the effect of a previous Install operation, or to cause an applied patch to be removed.

The Validate operation causes the software to be checked for its integrity, perhaps by invoking an algorithm generating a checksum, checking for viruses, etc.

Backing up software using the Backup operation is viewed as copying software, for example, for utilization should the current version fail. Backed-up software may be restored using the Restore operation.

Note that the optional Applied Patches attribute may need to be internally maintained by the managed system when it is not provided in the Managed Object class. The Applied Patches must be maintained if the Install or Revert services are supported (even if the Applied Patches attribute is not in the Managed Object class).

6.3.2 States of the Software Unit managed object

A Software Unit managed object can be in one of several internal states depending on the last operation that was performed on it. The possible states (and their meanings) are:

– Created – The delivery of the software product is not completed, however, some arbitrary resources on the managed system have been allocated to the Software Unit.

– Delivered – The Software Unit has been successfully delivered to the managed system.

– Installed – The software has been successfully installed on the managed system.

Table 1 maps the states for a Software Unit managed object to the state and status values defined in ITU-T Rec. X.731 [14].

Table 1/X.744.1 – Software Unit internal states

Software Unit State	**{Initialization Required} value of Procedural Status**	**{Not Installed} value of Availability Status**	**Resulting Operational State**
Created	Present	Present	Disabled
Delivered	Absent	Present	Disabled
Installed	Absent	Absent	Disabled or Enabled

For the Software Unit managed object, the Administrative State, Operational State, Procedural Status and Availability Status are mandatory state and status attributes, while the Usage State is an optional attribute.

The Created, Delivered and Installed internal states are mutually exclusive, that is, a Software Unit must be in only one of these states at a given time. Only the state transitions in Table 2 are permitted. Table 2 only addresses operations which can effect the states, other operations that do not cause a state change are excluded.

Table 2/X.744.1 – Software Unit internal state transition matrix

State Transition Matrix		Resulting State			
		(Non-existent)	**Created**	**Delivered**	**Installed**
State Initial	(Non-existent)	State change not possible	Create or local means[a)]	Create, local means or Deliver on Software Distributor[a)]	State change not possible
	Created	Delete	–	Local means[a)]	State change not possible
	Delivered	Delete	State change not possible	–	Install or local means[a)]
	Installed	Delete	State change not possible	Revert[a)]	Install, Revert or any operation that does not cause a state change (i.e., Backup, Validate and Restore)[a)]
a) Depending on object behaviour or local means.					

Independent of the Created, Delivered and Installed states are the Validating and Failed states.

Secondary states are the Validating and Failed states, which may be present in addition to the internal states. The Validating state maps to an Availability Status value which includes the value {In Test}. A Software Unit managed object may enter or exit the Validating state independently of the values of its other states. For example, a Software Unit managed object may be in the Validating state while failed (see Table 3).

Table 3/X.744.1 – Software Unit validating state

Software Unit State	{In Test} value of Availability Status	Resulting Operational State
Validating	Present	Disabled or Enabled

A Software Unit managed object in the Created, Delivered or Installed state may also be in the Failed state. The Failed state maps to an Availability Status value including the value {Failed}. Specific causes for entering and exiting the Failed state are a local matter. A Software Unit managed object may enter or exit the Failed state independently of the values of its other states. For examples, the side effect of a Validate operation may be a transition to the Failed state while in the Validating state, or a Software Unit may exit the Failed state as a result of a Revert operation (see Table 4).

Table 4/X.744.1 – Software Unit Failed state

Software Unit State	{Failed} value of Availability Status	Resulting Operational State
Failed	Present	Disabled

6.3.3 Operations on the Software Unit managed object

Associated with a Software Unit managed object are a number of operations which are used to make it change state.

These operations are:

– Backup – Causes the software to be backed up onto a target destination.
– Create – Causes a new Software Unit managed object to come into existence on the managed system.
– Delete – Causes the Software Unit managed object to be deleted on the managed and may additionally have the effect of deletion of associated resources.
– Install – Prepares the software for utilization.
– Restore – Causes backed-up software to be restored from a target destination.
– Revert – Causes the application of a patch or an installation to be reverted.
– Validate – Checks the integrity of the software.

The Create and Delete operations map onto the standard Create and Delete operations defined in ITU-T Rec. X.780 [24] while the remainder of these map onto methods. In addition, attribute values may be queried and modified using the Get and Set operations defined in ITU-T Rec. X.780.

6.3.3.1 Backup service

The backup service is used by a managing system to request performing a backup on the information represented by the target object instance (i.e., managed object representing the software being backed up).

This service uses the Backup operation in the Software Unit managed object class. The Backup operation can be performed independent of the state of the Software Unit managed object class. The Backup Destination parameter shall indicate the destination to which the information will be backed up. Possible destinations are:

– A local managed object – In this case, the Backup operation will be performed internally in the managed system. The information will be backed up to the supplied managed object instance.
– A remote system – In this case, the backup information will be transferred off-line to the remote system by some local means.

The following exceptions may be thrown:

– ITU-T Rec. X.780 [24] **Application Error**.
– **NOinformationBackupPackage** – If the Information Backup Package is not supported in this instance.
– **BackupSoftwareProcessingFailure** – Invalid Backup Destination parameter to the Backup operation.
– **ConcurrentOperationRequestFailure** – This Software Unit managed object class already has a pending Backup or Restore (or other) operation request. The criteria for whether concurrent requests are supported for a particular Software Unit managed object class is system specific.

Since a backup request may take significant time, the Backup operation will immediately return. A Backup Report notification will be issued following the completion of the target object backup.

6.3.3.2 Install service

The install service is used by a managing system to instruct a managed system to install a Delivered or Installed Software Unit object instance. If applicable, the install service will update the value of the Applied Patches attribute.

The Target Software attribute shall indicate the source of the software to be installed. The source must be unique, from an Applied Patches perspective. The source may be one or more of the following:

– Patch Id – System-specific identifier.

– Patch Pointer – Software Unit (or subclass) managed object class.

The Install operation will return the value of the Applied Patches attribute of the Software Unit object instance to which the service is directed.

The following exceptions may be thrown:

– ITU-T Rec. X.780 [24] **ApplicationError**.

– **NOinstallPackage** – If the Install Package is not supported in this instance.

– **InstallSoftwareProcessingFailure** – Invalid Install Info parameter to the Install operation.

– **OperationStateMismatch** – If the Software Unit cannot be installed due an invalid state condition. Only Software Unit managed objects in the Delivered or Installed state may be installed. In addition, the Operational State must be Enabled, the Administrative State must be Unlocked and the Usage State must be Active or Idle.

– **ConcurrentOperationRequestFailure** – This Software Unit managed object class already has a pending Install or Revert (or other) operation request. The criteria for whether concurrent requests are supported for a particular Software Unit managed object class is system specific.

6.3.3.3 Restore service

The restore service is used by a managing system to request performing a restore on the information represented by the target object instance. This will restore a previous backup.

This service uses the Restore operation in the Software Unit managed object class. The Restore operation can be performed independent of the state of the Software Unit managed object class. The Restore Source parameter shall indicate the destination to which the information will be restored from. Possible destinations are:

– A local managed object – A local managed object of the same class as the one this operation is applied to. In this case, the Restore operation will be performed internally in the managed system.

– A remote system – In this case, the restore information will be transferred off-line from the remote system by some local means.

The behaviour of the specific Software Unit instance or local means will determine the state of the Software Unit when it has been restored.

The following exceptions may be thrown:

– ITU-T Rec. X.780 [24] **ApplicationError**.

– **NOinformationRestorePackage** – If the Information Backup Package is not supported in this instance.

– **RestoreSoftwareProcessingFailure** – Invalid Resource Source parameter to the Restore operation.

– **ConcurrentOperationRequestFailure** – This Software Unit managed object class already has a pending Backup or Restore (or other) operation request. The criteria for whether concurrent requests are supported for a particular Software Unit managed object class is system specific.

Since a restore request may take significant time, the Restore operation will immediately return. A Restore Report notification will be issued following the completion of the target object restore.

6.3.3.4 Revert service

The revert service is used by a managing system (e.g., OS) to instruct a managed system to revert an applied patch or set of patches of the software represented by the Software Unit managed object.

This service uses the Revert operation in the Software Unit managed object class. The Revert Information parameter shall indicate one or more previously applied patches to which will be uninstalled. Each applied patch identifier is a choice of a system-specific identifier or a Software Unit object instance, depending on the values originally supplied in previous Install operation(s) on this Software Unit object instance.

The Revert operation will return the value of the Applied Patches attribute of the Software Unit object instance to which the service is directed.

If the revert service successfully reverts all patches that have thus far been installed (i.e., the Applied Patches attribute is empty), the Software Unit internal state will be changed from Installed to Delivered.

The following exceptions may be thrown:

– ITU-T Rec. X.780 [24] **Application Error**.

– **NOrevertPackage** – If the Revert Package is not supported in this instance.

– **OperationStateMismatch** – The Software Unit is in an invalid state for the Revert operation. To execute the Revert operation, the Software Unit must be in the Installed internal state, the Administrative State must be Unlocked, the Operational State must be Enabled and the Usage State must be Active or Idle.

– **RevertSoftwareProcessingFailure** – Invalid Revert Info parameter to the Revert operation.

– **ConcurrentOperationRequestFailure** – This Software Unit managed object class already has a pending Install or Revert (or other) operation request. The criteria for whether concurrent requests are supported for a particular Software Unit managed object class is system specific.

6.3.3.5 Validate service

This service uses the Validate operation in the Software Unit managed object class. The Validate Information parameter shall indicate what type of validation should be performed. Possible types are:

– Registered Validation – In this case, the validation information is provided via another specified managed object.

– Default Validation – In this case, the default validation for this specific managed object instance will be used.

The following exceptions may be thrown:

– ITU-T Rec. X.780 [24] **Application Error**.

– **NOvalidationPackage** – If the Validation Package is not supported in this instance.

– **ValidateSoftwareProcessingFailure** – One or more invalid arguments to the Validate operation.

– **OperationStateMismatch** – The Software Unit is in an invalid state for the Validate operation. To execute the Validate operation, the Software Unit must be in the Delivered or Installed internal state.

– **ConcurrentOperationRequestFailure** – This Software Unit managed object class already has a pending Validate (or other) operation request. The criteria for whether concurrent requests are supported for a particular Software Unit managed object class is system specific.

Since a validation request may take significant time, the Validation operation will immediately return. A Validate Report notification will be issued following the completzion of the target object validation.

6.3.4 Notifications on the Software Unit managed object

The Software Unit managed object has the following notifications, including those inherited from the Software managed object [2], (see Table 5):

Table 5/X.744.1 – Software Unit notifications

Notification	Reference	Conditional Package (if Conditional)
Attribute Value Change	[24]	"itut_m3120::attributeValueChangeNotificationPackage"
Backup Report	6.3.4.1	"itut_x744d1::informationBackupPackage" and/or "itut_x744d1::informationAutoBackupPackage"
Object Creation	[24]	"itut_m3120::createDeleteNotificationsPackage"
Object Deletion	[24]	"itut_m3120::createDeleteNotificationsPackage"
Processing Error	[24]	Mandatory
Restore Report	6.3.4.2	"itut_x744d1::informationRestorePackage" and/or "itut_x744d1::informationAutoRestorePackage"
State Change	[24]	"itut_m3120::stateChangeNotificationPackage"
Validate Report	6.3.4.3	"itut_x744d1::validationPackage"

Changes in the following attributes (when defined) will cause Attribute Value Change notifications (when supported) to be emitted:

– Affected Objects;
– Alarm Status;
– Applied Patches;
– Availability Status;
– Current Problem List;
– Future Auto Backup Destination;
– Future Auto Backup Trigger Threshold;
– Future Auto Restore Allowed;
– Future Auto Restore Source;
– Procedural Status;
– User Label;
– Version.

Changes in the following states (when defined) will cause State Change notifications (when supported) to be emitted:

– Operational State;
– Administrative State;
– Usage State.

6.3.4.1 Backup Report notification

The Backup Report notification is emitted to report a managed object backup. The backup may have been initiated automatically (according to criteria in the Future Auto Backup Trigger Threshold and Future Auto Backup Destination attributes), through management request (via the Backup operation) or initiated by the managed system.

The Backup Destination may be local (i.e., backup to another Software Unit object within the local managed system) or off-line to a remote system by using a particular file transfer protocol (e.g., FTAM). The result of the backup will be reported in this notification.

6.3.4.2 Restore Report notification

The Restore Report notification is emitted to report a restore of a managed object from a previous backup. The restore may have been initiated automatically (according to the Future Auto Restore Source Type and Future Auto Restore Allowed Type attributes and system-specific criteria), through management request (via the Restore operation) or initiated by the managed system.

The Restore Source may be local (i.e., restore from another Software Unit object within the local managed system) or off-line from a remote system by using a particular file transfer protocol (e.g., FTAM). The result of the restore will be reported in this notification.

6.3.4.3 Validate Report notification

The Validate Report notification is emitted to report the results of a Validate operation.

6.4 Executable Software managed object

The Executable Software managed object class is a subclass of the Software Unit managed object class (see 6.3.1) with additional characteristics to describe its functionality as executable. Executable software represents software that may be executed in some way such as via local means or remotely via the Execute operation. It may be possible to cause Executable Software to be executed by management command; however, it may only be possible to execute software by local action.

Although out of scope in this Recommendation, the model does not preclude subclasses of Executable Software from including information such as whether software can have single or multiple users, under what conditions software is active or busy, or whether a maximum number of users are allowed.

6.4.1 Additional states for the Executable Software managed object

The Usage State, defined in ITU-T Rec. X.731 [14], reflects whether the Executable Software is currently being utilized. For the Executable Software managed object class, the Usage State is a mandatory attribute in addition to those already in the Software Unit managed object (see 6.3.2). In the Executable Software managed object class, the Usage State values of Idle, Active and Busy are permitted.

The Usage State value of Idle means that there are no active executions. What the values of Active and Busy mean are implementation specific and are out of the scope of this Recommendation. For example, the value of Active could mean that there exist some running execution, while the value of Busy means that the program has reached its maximum capacity and any further request of execution (by using the Execute Program service or by some other local means) may be denied or queued for later execution. Specializations may want to include a threshold for the number of users required to reach maximum capacity.

6.4.2 Additional operations for the Executable Software managed object

The following are operations on the Executable Software managed object in addition to those defined for the Software Unit managed object (see 6.3.3):

– Execute Program – This causes the software managed object to be executed.

The Execute Program operation is used to initiate the execution of software program represented by the Executable Software. The Executable Software must be installed in order for it to be executed.

The behaviour of the specific Executable Software instance or local means will determine the state of the Executable Software when it has been executed.

6.4.2.1 Execute Program service

The Execute Program service is used by a managing system to initiate the execution of a program represented by an Executable Software object.

The Additional Information parameter shall indicate the parameters used in the execution of the software.

Following a successful initiation, the Execute Program Reply will contain:

– Process Id.

– Process Owner.

– Initial start time.

– supplied Additional Information parameters.

The following exceptions may be thrown:

– ITU-T Rec. X.780 [24] **ApplicationError**.

– **NOexecuteProgramPackage** – If the Execute Program Package is not supported in this instance.

– **OperationStateMismatch** – The Executable Software is in an invalid state for the Execute Program operation. To execute the Execute Program operation, the Executable Software must be in the Installed internal state, the Administrative State must be Unlocked, the Operational State must be Enabled and the Usage State must either be Active or Idle.

– **ExecuteProgramSoftwareProcessingFailure** – One or more invalid arguments to the Execute Program operation.

6.5 Software Distributor managed object

A Software Distributor managed object is a static object which represents delivery mechanism(s) of a managed system. It is a managed object which distributes software to the target managed system when it receives a Deliver operation from the managing system. The parameters of the Deliver operation may be used to indicate the set of software to be delivered, the target destination for delivery, and the choice of transfer mechanism. Although this object class may be used to initiate delivery via a variety of transfer mechanisms, it does not model any such transfer mechanisms. These models are left for specializations.

This managed object emits a Deliver Result notification with results of the distribution when the distribution is completed.

6.5.1 Operations on the Software Distributor managed object

The following are operations on the Software Distributor managed object:

– Create – Causes a new Software Distributor managed object to come into existence. An Object Creation notification will be emitted.

– Deliver – Causes the Software Distributor managed object to cause creation of the specified software (by a method out of scope of this Recommendation) onto the target managed system and causes any associated resources which are to be associated with those Software Unit managed objects to be created onto the target managed as a side effect.

– Delete – Causes the Software Distributor managed object to be deleted on the managed system. An Object Deletion notification will be emitted.

6.5.1.1 Deliver service

The deliver service is used by a managing system to request distribution of software or a set of software. The Deliver operation information identifies the software that is to be distributed. The result of a Deliver operation is that a copy of the target software items is delivered to the target system in the Delivered internal state.

Packaging of the software and the choice of transfer mechanism is a local matter and outside the scope of this Recommendation. For example, this information may be pre-configured or specified in the Deliver operation along with any other associated information.

The result of successful completion is that the software to be distributed is copied to the target system; this may have the effect of Software Unit and/or Executable Software objects being created. After command completion, a Deliver Result notification is emitted.

The following parameters are used with the Deliver operation:

– Deliver Id – This optional parameter indicates a unique identifier for this Deliver operation.

– Target Software – This indicates the source of the software to be delivered.

– Target System – This indicates the optional target destination for the software to be delivered. If this does not indicate a target destination, the system uses local means to determine the target destination.

– Transfer Information – Application-specific transfer mechanism.

– Additional Information – Additional application-specific information.

The following exceptions may be thrown:

– ITU-T Rec. X.780 [24] **ApplicationError**.

– **OperationStateMismatch** – The Software Distributor is in an invalid state for the Deliver operation. To execute the Deliver operation, the Software Distributor must be in the Unlocked Administrative State and the Enabled Operational State.

– **DeliverSoftwareProcessingFailure** – One or more invalid arguments to the Deliver operation.

6.5.2 Notifications on the Software Distributor managed object

The Software Distributor managed object has the following notifications (see Table 6):

Table 6/X.744.1 – Software Distributor notifications

Notification	Reference	Conditional Package (if Conditional)
Deliver Result	6.5.2.1	Mandatory
Object Creation	[24]	Mandatory
Object Deletion	[24]	Mandatory
State Change	[24]	Mandatory

Changes in the following states will cause State Change notifications to be emitted:

– Operational State;

– Administrative State.

6.5.2.1 Deliver Result notification

The Deliver Result notification is emitted from the managed object when the Deliver operation is completed. It contains the final results of the operation and may indicate either a pass or fail condition.

6.6 Use of Universal Identifiers (UIDs)

This Recommendation makes use of universal identifiers (UIDs), as defined in ITU-T Rec. X.780 [24]. Universal identifiers allow for local extensions of both constant and data values. Examples for where this may be used are to allow locally-defined arguments to an Execute Program operation and to extend the listed File Type file types.

Universal identifiers make use of the UID Type, Management Extension Type and Additional Information Set Type attribute types. The IDL syntax for these types (from ITU-T Rec. X.780) are as follows:

```
struct UIDType {
  string moduleName; // module where value is defined
  short value; // constant within the module
};

struct ManagementExtensionType {
  UIDType id; // identifies the type of info
  any info; // type will depend on id
};

typedef sequence <ManagementExtensionType> AdditionalInformationSetType;
```

To make a local extension of a universal identifier, UID constant modules must be defined in new IDL. The UID constant modules will contain a constant for each supported constant (with UID Type) or data type (with Management Extension Type and Additional Information Set Type). Examples of UID constant modules used in this model can be found in 9.19 and 9.20.

For a complete description of universal identifiers, see ITU-T Rec. X.780.

6.7 Relationships

A number of relationships between software managed objects and also between software managed objects and other managed objects have been identified. These are:

– Dependency – This relationship may be used to model the fact that one software managed object is dependent in some way on the presence of another software managed object. Such a relationship can be used to model patching.
– Configuration – This relationship may be used to model the fact that one Software Unit managed object may affect the behaviour of another Software Unit managed object. For example an additional font could be modeled as a configuration relationship. This relationship can also be used to model enhancements and patching.
– Utilization – This relationship may be used to show what other managed objects utilize software managed objects. Such other managed objects are likely to be those representing processes running on the managed system.

Failure of a managed object may cause any managed objects which are dependent upon it to also fail. However, it will not cause managed objects with a configuration relationship to fail (though their behaviour may change). The detection of faulty software may result in the requirement to deliver and install a new copy of the software or to deliver and install an updated version.

However, specification of any of these relationships is outside the scope of this Recommendation, as the relationships are application dependent.

7 Relationship with other functions

The following functions are provided by other Systems Management Functions:

– Performance of software, covered by the ITU-T Rec. Q.822.1 [8];
– Software audit trailing, covered by the Security Audit Trail Function (see ITU-T Rec. X.740 [15]);
– Support for software security, covered by Objects and Attributes for Access Control (see ITU-T Rec. X.741 [16]);
– Accounting of software usage, covered by the Usage Metering Function for Accounting Purposes (see ITU-T Rec. X.742 [17]);
– Software Testing (includes Installation of the test environment, Test run of the software, Setting breakpoints, and suspending and resuming the software test environment), covered by the Test Management Function (see ITU-T Rec. X.745 [22]);
– Scheduling of software functions and operations, covered by the Scheduling Function (see ITU-T Rec. X.746 [23]).

8 Compliance and conformance

This clause defines the criteria that must be met by other standards claiming compliance to this Recommendation and the functions that must be implemented by systems claiming conformance to this Recommendation.

8.1 System conformance

8.1.1 Conformance points

This clause describes the conformance points that must be supported by systems claiming conformance to these specifications:

1) An implementation claiming conformance to these requirements must:
   - support either:
     – the Basic Conformance Profile in ITU-T Rec. Q.816 [3]. In this case, each managed object instance will be an instantiated CORBA object; or
     – the Basic Conformance Profile in ITU-T Rec. Q.816.1 [7]. In this case, each supported managed object class will have an instantiated Facade CORBA object (see ITU-T Recs Q.816.1 and X.780.1 [28]).
   - support either:
     – the Software Unit requirements (without Executable Software or Software Distributor support); or
     – the Executable Software requirements (without Software Unit or Software Distributor support); or
     – the Software Distributor requirements (without Software Unit or Executable Software support). This may occur before software is delivered; or
     – both the Software Unit and Executable Software requirements (without the Software Distributor support); or
     – both the Software Unit and Software Distributor requirements (without the Executable Software support); or
     – both the Executable Software and Software Distributor requirements (without the Software Unit support); or
     – the Software Unit, Executable Software and Software Distributor requirements.
   - use the IDL listed in clause 9.
2) An implementation claiming conformance to the Software Unit requirements must:
   - support the Software Unit managed object specified in 6.3.
   - support the creation of at least one managed object of the Software Unit managed object class.
3) An implementation claiming conformance to the Executable Software requirements must:
   - support the Executable Software managed object specified in 6.4.
   - support the creation of at least one managed object of Executable Software managed object class.
4) An implementation claiming conformance to the Software Distributor requirements must:
   - support the Software Distributor managed object specified in 6.5.
   - support the creation of at least one managed object of Software Distributor managed object class.

8.2 Conformance statement guidelines

The users of this framework must be careful when writing conformance statements. Because IDL modules are being used as name spaces, they may, as allowed by OMG IDL rules, be split across files. Thus, when a module is extended its name won't change. Instead, a new IDL file will simply be added. Simply stating the name of a module in a conformance statement, therefore, will not suffice to identify a set of IDL interfaces. The conformance statement must identify a document and year of publication to make sure the right version of IDL is identified.

9 ITU-T Rec. X.744.1 IDL listing

```
#ifndef _itut_x744_1_idl_
#define _itut_x744_1_idl_

#include <itut_x780.idl>
#include <itut_x780_1.idl>
#include <itut_x780ct.idl>
#include <itut_m3120.idl>

#pragma prefix "itu.int"

/**
This IDL code (beginning with the line "#ifndef … " through the end of this
section) is intended to be stored in a file named “itut_x744_1.idl” located in
the search path used by the IDL compiler on your system. A compiler supporting
the CORBA version specified in ITU-T Rec. Q.816 must be used.
*/

/**
This module, itut_x744d1, contains the IDL interface definition for ITU-T Rec.
X.744. The IDL definitions in this file are the object interfaces.
*/

module itut_x744d1
{

/**
```

9.1 Imports

```
*/
     /**
     Types imported from ITU-T Rec. X.780
     */

     typedef itut_x780::AdditionalInformationSetType AdditionalInformationSetType;
     typedef itut_x780::AdministrativeStateType AdministrativeStateType;
     typedef itut_x780::ApplicationErrorInfoType ApplicationErrorInfoType;
     typedef itut_x780::AvailabilityStatusSetType AvailabilityStatusSetType;
     typedef itut_x780::DeletePolicyType DeletePolicyType;
     typedef itut_x780::ExternalTimeType ExternalTimeType;
     typedef itut_x780::GeneralizedTimeType GeneralizedTimeType;
     typedef itut_x780::Istring Istring;
     typedef itut_x780::ManagementExtensionType ManagementExtensionType;
     typedef itut_x780::MONameType MONameType;
     typedef itut_x780::NameBindingType NameBindingType;
     typedef itut_x780::NullType NullType;
     typedef itut_x780::OperationalStateType OperationalStateType;
     typedef itut_x780::ProceduralStatusSetType ProceduralStatusSetType;
     typedef itut_x780::StringSetType StringSetType;
     typedef itut_x780::UIDType UIDType;
     typedef itut_x780::UsageStateType UsageStateType;

     /**
     Types imported from ITU-T Rec. M.3120
     */
     typedef itut_m3120::AlarmStatusType AlarmStatusType;
     typedef itut_m3120::AlarmSeverityAssignmentProfileNameType
             AlarmSeverityAssignmentProfileNameType;
     typedef itut_m3120::ArcProbableCauseSetType ArcProbableCauseSetType;
     typedef itut_m3120::ArcIntervalProfileNameType ArcIntervalProfileNameType;
     typedef itut_m3120::ArcTimeType ArcTimeType;

/**
```

9.2 Forward declarations

```
*/
     /**
     Interface forward declarations
     */

     interface ExecutableSoftware;
     interface ExecutableSoftware_F;
     interface ExecutableSoftwareFactory;
     interface SoftwareDistributor;
     interface SoftwareDistributor_F;
     interface SoftwareDistributorFactory;
     interface SoftwareUnit;
     interface SoftwareUnit_F;
     interface SoftwareUnitFactory;

     /**
     valuetype forward declarations
     */

     valuetype ExecutableSoftwareValueType;
     valuetype SoftwareDistributorValueType;
     valuetype SoftwareUnitValueType;

/**
```

9.3 Structures and typedefs

```
*/

     /**
     Patch ::= CHOICE {
       patchId GraphicString, -- system specific identifier --
       patchPointer ObjectInstance } -- of Software Unit object class –
     AppliedPatches ::= SEQUENCE OF Patch
     */

     enum PatchChoice
     {
       patchIdChoice, // system specific identifier
       patchPointerChoice // of Software Unit (or Executable Software) object
                          // class
     };

     union PatchType switch (PatchChoice)
     {
       case patchIdChoice:
          Istring patchId;
       case patchPointerChoice:
          MONameType patchPointer;
     };
```

```
/**
Applied Patches attribute type
*/

typedef sequence <PatchType> AppliedPatchesSeqType;

/**
BackupDestination ::= CHOICE {
  localObject ObjectInstance,
  inLine NULL, -- in-line in the notification in additionalInfo --
  offLine GraphicString -- remote system by, e.g., FTAM --}

The inLine choice is not supported in ITU-T Rec. X.744.1
*/

enum BackupDestinationChoice
{
  localObjectChoice,
  offLineChoice // remote system by, e.g., FTAM
};

union BackupDestinationType switch (BackupDestinationChoice)
{
  case localObjectChoice:
     MONameType localObject;
  case offLineChoice:
     Istring offLine;
};

/**
CheckSum ::= BIT STRING

Check Sum attribute type. Algorithm for calculating check sum is locally
determined
*/

typedef long CheckSumType;

/**
Date Of Creation attribute type
*/

typedef GeneralizedTimeType DateOfCreationType;

/**
Date ::= CHOICE {
  time   GeneralizedTime ,
  noSuchInformationNULL}
*/

enum DateChoice
{
  timeChoice,
  noSuchInformationChoice
};

union DateType switch (DateChoice)
{
  case timeChoice:
     GeneralizedTimeType time;
  case noSuchInformationChoice:
     NullType noInformation;
};
```

```
/**
File Location attribute type
*/

typedef sequence <Istring> FileLocationSetType;

/**
FileType ::= INTEGER{
  unstructuredText (0), -- FTAM-1
  unstructuredBinary (1), -- FTAM-3
  blockSpecial (2)}

File Type has been converted to UIDType, originally based on FileTypeConst
module. This allows applications to add their own file types

File Type attribute type
*/

typedef UIDType FileTypeType;

/**
Future Auto Backup Trigger Threshold attribute type

Note that the float type is used to match ITU-T Rec. Q.822.1 threshold
types, but this is not a true Q.822.1 threshold
*/

typedef float FutureAutoBackupTriggerThresholdType;

/**
Future Auto Restore Allowed attribute type – TRUE means they are allowed
*/

typedef boolean FutureAutoRestoreAllowedType;

/**
AutoRestoreSource ::= CHOICE {
  localObject ObjectInstance,
  remoteSystem GraphicString -- off-line from remote system
}
*/

enum AutoRestoreSourceChoice
{
  autoRestoreSourceLocalObjectChoice,
  autoRestoreSourceRemoteSystemChoice // off-line from remote system
};

union AutoRestoreSourceType switch (AutoRestoreSourceChoice)
{
  case autoRestoreSourceLocalObjectChoice:
     MONameType localObject;
  case autoRestoreSourceRemoteSystemChoice:
     Istring offLine;
};

/**
Future Auto Restore Source attribute type
*/

typedef AutoRestoreSourceType FutureAutoRestoreSourceType;
```

```
/**
Identity Of Creator attribute type
*/

typedef Istring IdentityOfCreatorType;

/**
Identity Of Last Modifier attribute type
*/

typedef Istring IdentityOfLastModifierType;

/**
InformationSize ::=     CHOICE {
  numberOfBits [0] INTEGER,
  numberOfBytes    [1] INTEGER}
*/

enum InformationSizeChoice
{
  numberOfBitsChoice,
  numberOfBytesChoice
};

union InformationSizeType switch (InformationSizeChoice)
{
  case numberOfBitsChoice:
     long bits;
  case numberOfBytesChoice:
     long bytes;
};

/**
LastBackupDestination ::= CHOICE {
  notBackedUp NULL,
  localObject ObjectInstance,
  managingSystem AE-title,
  remoteSystem GraphicString}

The inLine choice is not supported in ITU-T Rec. X.744.1
*/

enum LastBackupDestinationChoice
{
  lastBackupDestinationLocalObjectChoice,
  lastBackupDestinationOffLineChoice,
  lastBackupDestinationNotBackedUpChoice
};

/**
Last Backup Destination attribute type
*/

union LastBackupDestinationType switch (LastBackupDestinationChoice)
{
  case lastBackupDestinationLocalObjectChoice:
     MONameType localObject;
  case lastBackupDestinationOffLineChoice:
     Istring offLine;
  case lastBackupDestinationNotBackedUpChoice:
     NullType noInformation;
};
```

```
/**
LastRestoreSource ::= CHOICE {
  notRestored NULL,
  localObject ObjectInstance,
  managingSystem AE-title,
  remoteSystem GraphicString}

The inline choice is not supported in ITU-T Rec. X.744.1
*/

enum LastRestoreSourceChoice
{
  lastRestoreSourceLocalObjectChoice,
  lastRestoreSourceOffLineChoice,
  lastRestoreSourceNotRestoredChoice
};

/**
Last Restore Source attribute type
*/

union LastRestoreSourceType switch (LastRestoreSourceChoice)
{
  case lastRestoreSourceLocalObjectChoice:
     MONameType localObject;
  case lastRestoreSourceOffLineChoice:
     Istring offLine; // off-line from remote system
  case lastRestoreSourceNotRestoredChoice:
     NullType noInformation;
};

/**
Note Field attribute type
*/

typedef Istring NoteFieldType;

/**
Date Delivered attribute type
*/

typedef DateType DateDeliveredType;

/**
Date Installed attribute type
*/

typedef DateType DateInstalledType;

/**
Date Of Last Modification attribute type
*/

typedef DateType DateOfLastModificationType;

/**
File Size attribute type
*/

typedef InformationSizeType FileSizeType;
```

```
/**
Future Auto Backup Destination attribute type
*/

typedef BackupDestinationType FutureAutoBackupDestinationType;

/**
Last Backup Time attribute type
*/

typedef DateType LastBackupTimeType;

/**
Last Restore Time attribute type
*/

typedef DateType LastRestoreTimeType;

/**
BackupResult ::= CHOICE {
  inLine [0]  CHOICE {
     successBIT STRING,
     fail-pduSizeLimitation  [3]  NULL,
     fail-securityLicensing  [4]  NULL,
     fail-unknown  [5]  NULL},
  local [1] SEQUENCE {
     destination ObjectInstance, -- in the managed system --
     success BOOLEAN -- TRUE for success --
     },
  offLine [2] SEQUENCE {
     destination GraphicString, --the remote system
     result CHOICE {
      success  [6]  NULL,
      fail-securityLicensing [7]  NULL,
      fail-unknown  [8]  NULL}
}}

The inLine choice is not supported in ITU-T Rec. X.744.1
*/

enum BackupResultChoice
{
  backupResultFailureChoice, // some type of error occurred while backing
                             // up the object
  backupResultLocalChoice,
  backupResultOffLineChoice
};

union BackupResultType switch (BackupResultChoice)
{
  case backupResultFailureChoice:
     ApplicationErrorInfoType error; // from ITU-T Rec. X.780
  case backupResultLocalChoice:
     MONameType localObject; // in the managed system
  case backupResultOffLineChoice:
     Istring offLine; // in the managed system
};
```

```
/**
DeliverId ::= CHOICE {
  globalValue OBJECT IDENTIFIER,
  localValue  INTEGER}
*/

typedef long DeliverIdType;

/**
DeliverIdTypeOpt is an optional type. If the discriminator is true the
value is present, otherwise the value is NullType.
*/

union DeliverIdTypeOpt switch (boolean)
{
  case TRUE:
     DeliverIdType value;
  case FALSE:
     NullType noInformation;
};

/**
DeliverResult ::= INTEGER {
  pass (0),
  communicationError (1),
  equipmentError (2),
  qosError (3),
  accessDenied (4),
  notFound (5),
  insufficientSpace (6),
  alreadyDelivered (7),
  inProgress (8),
  unknown (9) }

Deliver Result has been converted to UIDType, originally based on
DeliverResultConst module. This allows applications to add their own
deliver results.
*/

typedef UIDType DeliverResultType;

/**
Destination ::= CHOICE {
  single  AE-title,
  multiple  SET OF AE-title}
-- Note that the syntax of AE-title to be used is from ITU-T Rec. X.227 |
-- ISO/IEC 8650-1 Amendment 1 and not "ANY".
*/

typedef sequence <MONameType> MONameSetType;

enum DestinationChoice
{
  singleChoice,
  multipleChoice
};

union DestinationType switch (DestinationChoice)
{
  case singleChoice:
     MONameType single;
  case multipleChoice:
     MONameSetType multipleValues;
};
```

```
/**
ExecuteProgramReply ::= SEQUENCE {
  processId INTEGER,
  processOwner Identity,
  startTime GeneralizedTime,
  additionalInfo   SET OF ManagementExtension OPTIONAL }
*/

typedef Istring IdentityType;

struct ExecuteProgramReplyType
{
  long processId;
  IdentityType processOwner;
  GeneralizedTimeType startTime;
  AdditionalInformationSetType additionalInfo;
};

typedef AppliedPatchesSeqType InstallReplyType;

/**
Who generated the request?
*/

enum RequestType
{
  automaticRequest,
  managementRequest,    // i.e., method execution
  managedSystemRequest
};

/**
What were the results of the restore operation?
*/

enum RestoreResultChoice
{
  restoreResultFailureChoice, // some type of error occurred while
                              // restoring the object
  restoreResultLocalChoice,
  restoreResultOffLineChoice
};

union RestoreResultType switch (RestoreResultChoice)
{
  case restoreResultFailureChoice:
     ApplicationErrorInfoType error; // from ITU-T Rec. X.780
  case restoreResultLocalChoice:
     MONameType localObject; // in the managed system
  case restoreResultOffLineChoice:
     Istring offLine; // in the managed system
};
```

```
/**
RestoreSource ::= CHOICE {
  localObject ObjectInstance,
  inLine BIT STRING,
  offLine GraphicString
     -- remote system via some other transfer protocol, e.g., FTAM --
}

The inline choice is not supported in ITU-T Rec. X.744.1
*/

enum RestoreSourceChoice
{
  restoreSourceLocalObjectChoice,
  restoreSourceOffLineChoice
     // remote system via some other transfer protocol, e.g., FTAM
};

union RestoreSourceType switch (RestoreSourceChoice)
{
  case restoreSourceLocalObjectChoice:
     MONameType localObject;
  case restoreSourceOffLineChoice:
     Istring offLine;
};

/**
RevertInfo ::= SEQUENCE OF CHOICE {
  patchId GraphicString, -- system specific identifier --
  patchPointer ObjectInstance } -- Executable Software object class –
*/

enum RevertChoice
{
  revertPatchIdChoice, // system specific identifier
  revertPatchPointerChoice // Software Unit (or subclass) object class
};

union RevertType switch (RevertChoice)
{
  case revertPatchIdChoice:
     Istring patchId;
  case revertPatchPointerChoice:
     MONameType patchPointer;
};

typedef sequence <RevertType> RevertInfoSetType;

/**
RevertReply ::= SEQUENCE {
  revertedPatches  [0] AppliedPatches,
  additionalInfo   [1] SET OF ManagementExtension OPTIONAL }
*/

typedef AppliedPatchesSeqType RevertReplyType;

/**
DistributedSoftware ::= CHOICE {
  distibutedSoftwareId  GraphicString,
  distributedSoftwarePointer ObjectInstance }
*/
```

```
enum DistributedSoftwareChoice
{
  distributedSoftwareIdChoice,
  distributedSoftwarePointerChoice
};

union DistributedSoftwareType switch (DistributedSoftwareChoice)
{
  case distributedSoftwareIdChoice:
     Istring patchId; // system specific identifier
  case distributedSoftwarePointerChoice:
     MONameType patchPointer; // SoftwareUnit (or subclass) object class
};

typedef sequence <DistributedSoftwareType> TargetSoftwareSetType;

/**
Note that AdditionalInformationSetType is a set of ManagementExtensionType
*/

typedef ManagementExtensionType TransferInfoType;

/**
ValidateInfo ::= CHOICE {
  instanceDefaultValidationType  [0] NULL, -- local matter --
  registeredValidationType  [1] OBJECT IDENTIFIER }
*/

enum ValidateInfoChoice
{
  registeredValidationTypeChoice,
  instanceDefaultValidationTypeChoice // local matter
};

union ValidateInfoType switch (ValidateInfoChoice)
{
  case registeredValidationTypeChoice:
     MONameType instanceDefaultValidationType;
  case instanceDefaultValidationTypeChoice:
     NullType noInformation;
};

/**
ValidateReply ::= CHOICE {
  validationTerminated  [0] NULL,
  passValidation   [1] NULL,
  passValidationWithResult   [2] SET OF ManagementExtension,
  failValidation   [3] NULL,
  failValidationWithResult   [4] SET OF ManagementExtension }

The Terminate Validation operation is not supported in ITU-T Rec.
X.744.1
*/

enum ValidateResultChoice
{
  passValidationWithResultChoice,
  failValidationWithResultChoice,
  passValidationChoice,
  failValidationChoice
};
```

```
union ValidateResultType switch (ValidateResultChoice)
{
  case passValidationWithResultChoice:
     AdditionalInformationSetType passValidationWithResult;
  case failValidationWithResultChoice:
     AdditionalInformationSetType failValidationWithResult;
  case failValidationChoice:
     ApplicationErrorInfoType error; // from ITU-T Rec. X.780
  case passValidationChoice:
     NullType noInformation;
};

/**
BackupReply ::= SEQUENCE {
  reply  [0] CHOICE {
     successNULL, -- for local or off-line backup
     inLine BIT STRING },
  additionalInfo   [1] SET OF ManagementExtension OPTIONAL }

In ITU-T Rec. X.744.1, the Backup operation issues a notification of
the results instead of returning the results
*/

/**
TerminateValidationInfo ::= ENUMERATED {
  cancel (0), -- discard the result of the partial audit --
  truncate (1) } -- report the result of the partially completed audit –

TerminateValidationReply ::= CHOICE {
  noOutStandingValidation    [0] NULL,
  validationCancelled   [1] NULL,
  resultOfPartialValidation  [2] ValidateReply}

The Terminate Validation operation is not supported in ITU-T Rec.
X.744.1
*/

/**
BackupDestinationTypeOpt is an optional type. If the discriminator is true
the value is present, otherwise the value is NullType.
*/

union BackupDestinationTypeOpt switch (boolean)
{
  case TRUE:
     BackupDestinationType value;
  case FALSE:
     NullType noInformation;
};

/**
RestoreSourceTypeOpt is an optional type. If the discriminator is true the
value is present, otherwise the value is NullType.
*/

union RestoreSourceTypeOpt switch (boolean)
{
  case TRUE:
     RestoreSourceType value;
  case FALSE:
     NullType noInformation;
};
```

```
/**
TargetSoftwareSetTypeOpt is an optional type. If the discriminator is true
the value is present, otherwise the value is NullType.
*/

union TargetSoftwareSetTypeOpt switch (boolean)
{
  case TRUE:
     TargetSoftwareSetType value;
  case FALSE:
     NullType noInformation;
};

/**
RevertInfoSetTypeOpt is an optional type. If the discriminator is true the
value is present, otherwise the value is NullType.
*/

union RevertInfoSetTypeOpt switch (boolean)
{
  case TRUE:
     RevertInfoSetType value;
  case FALSE:
     NullType noInformation;
};

/**
ValidateInfoTypeOpt is an optional type. If the discriminator is true the
value is present, otherwise the value is NullType.
*/

union ValidateInfoTypeOpt switch (boolean)
{
  case TRUE:
     ValidateInfoType value;
  case FALSE:
     NullType noInformation;
};

struct ValidateSoftwareProcessingErrorType
{
  ValidateInfoTypeOpt validateInfo;
};

/**
DestinationTypeOpt is an optional type. If the discriminator is true the
value is present, otherwise the value is NullType.
*/

union DestinationTypeOpt switch (boolean)
{
  case TRUE:
     DestinationType value;
  case FALSE:
     NullType noInformation;
};

/**
TransferInfoTypeOpt is an optional type. If the discriminator is true the
value is present, otherwise the value is NullType.
*/
```

```
    union TransferInfoTypeOpt switch (boolean)
    {
      case TRUE:
         TransferInfoType value;
      case FALSE:
         NullType noInformation;
    };

    struct DeliverSoftwareProcessingErrorType
    {
      DeliverIdTypeOpt deliverId;
      TargetSoftwareSetTypeOpt targetSoftware;
      DestinationTypeOpt targetSystem;
      TransferInfoTypeOpt transferInfo;
      AdditionalInformationSetType additionalInfo;
    };

    const string administrativeStatePackage =
      "itut_x744d1::administrativeStatePackage";
    const string appliedPatchPackage = "itut_x744d1::appliedPatchPackage";
    const string checkSumPackage = "itut_x744d1::checkSumPackage";
    const string createDeleteNotificationsPackage =
      "itut_x744d1::createDeleteNotificationsPackage";
    const string executeProgramPackage = "itut_x744d1::executeProgramPackage";
    const string fileInformationPackage =
      "itut_x744d1::fileInformationPackage";
    const string filePackage = "itut_x744d1::filePackage";
    const string informationAutoBackupPackage =
      "itut_x744d1::informationAutoBackupPackage";
    const string informationAutoRestorePackage =
      "itut_x744d1::informationAutoRestorePackage";
    const string informationBackupPackage =
      "itut_x744d1::informationBackupPackage";
    const string informationRestorePackage =
      "itut_x744d1::informationRestorePackage";
    const string installPackage = "itut_x744d1::installPackage";
    const string noteFieldPackage = "itut_x744d1::noteFieldPackage";
    const string revertPackage = "itut_x744d1::revertPackage";
    const string stateChangeNotificationPackage =
      "itut_x744d1::stateChangeNotificationPackage";
    const string usageStatePackage = "itut_x744d1::usageStatePackage";
    const string validationPackage = "itut_x744d1::validationPackage";

/**
```

9.4 Exceptions

```
*/

    exception NOadministrativeStatePackage {};
    exception NOappliedPatchPackage {};
    exception NOcheckSumPackage {};
    exception NOexecuteProgramPackage {};
    exception NOfileInformationPackage {};
    exception NOfilePackage {};
    exception NOinformationAutoBackupPackage {};
    exception NOinformationAutoRestorePackage {};
    exception NOinformationBackupPackage {};
    exception NOinformationRestorePackage {};
    exception NOinstallPackage {};
    exception NOnoteFieldPackage {};
    exception NOrevertPackage {};
    exception NOusageStatePackage {};
    exception NOvalidationPackage {};
```

```
     exception BackupSoftwareProcessingFailure
     {
       BackupDestinationTypeOpt backupDestionation;
     };

     /**
     Used by various methods to indicate that this operation failed due to other
     pending or executing operations
     */

     exception ConcurrentOperationRequestFailure {};

     exception RestoreSoftwareProcessingFailure
     {
       RestoreSourceTypeOpt attributes;
     };

     exception InstallSoftwareProcessingFailure
     {
       TargetSoftwareSetTypeOpt attributes;
     };

     exception RevertSoftwareProcessingFailure
     {
       RevertInfoSetTypeOpt attributes;
     };

     exception ValidateSoftwareProcessingFailure
     {
       ValidateSoftwareProcessingErrorType attributes;
     };

     exception ExecuteProgramSoftwareProcessingFailure
     {
       AdditionalInformationSetType additionalInfo;
     };

     exception DeliverSoftwareProcessingFailure
     {
       DeliverSoftwareProcessingErrorType attributes;
     };

     /**
     Used by various methods to indicate the operation cannot be performed due
     to a state condition that is invalid for this operation
     */

     exception OperationStateMismatch {};

/**
```

9.5 Software unit

```
*/
     /**
     This valuetype is used to retrieve all attributes
     */

     valuetype SoftwareUnitValueType : truncatable itut_m3120::SoftwareValueType
     {
       public AvailabilityStatusSetType availabilityStatus;
          // GET
       public ProceduralStatusSetType proceduralStatus;
```

```
   // GET
public AppliedPatchesSeqType appliedPatches;
   // GET
   // appliedPatchPackage
public CheckSumType checkSum;
   // GET
   // checkSumPackage
public DateOfCreationType dateOfCreation;
   // GET
   // fileInformationPackage
public IdentityOfCreatorType identityOfCreator;
   // GET
   // fileInformationPackage
public DateOfLastModificationType dateOfLastModification;
   // GET
   // fileInformationPackage
public IdentityOfLastModifierType identityOfLastModifier;
   // GET
   // fileInformationPackage
public DateDeliveredType dateDelivered;
   // GET
   // fileInformationPackage
public DateInstalledType dateInstalled;
   // GET
   // fileInformationPackage
public FileLocationSetType fileLocation;
   // GET
   // filePackage
public FileSizeType fileSize;
   // GET
   // filePackage
public FileTypeType fileType;
   // GET
   // filePackage
public FutureAutoBackupTriggerThresholdType
         futureAutoBackupTriggerThreshold;
   // GET-REPLACE
   // informationAutoBackupPackage
public FutureAutoBackupDestinationType futureAutoBackupDestination;
   // GET-REPLACE
   // informationAutoBackupPackage
public FutureAutoRestoreSourceType futureAutoRestoreSource;
   // GET-REPLACE
   // informationAutoRestorePackage
public FutureAutoRestoreAllowedType futureAutoRestoreAllowed;
   // GET-REPLACE
   // informationAutoRestorePackage
public LastBackupTimeType lastBackupTime;
   // GET
   // informationBackupPackage
public LastBackupDestinationType lastBackupDestination;
   // GET
   // informationBackupPackage
public LastRestoreTimeType lastRestoreTime;
   // GET
   // informationRestorePackage
public LastRestoreSourceType lastRestoreSource;
   // GET
   // informationRestorePackage
public NoteFieldType noteField;
   // GET-REPLACE
   // noteFieldPackage
```

```
    public UsageStateType usageState;
       // GET
       // usageStatePackage

  }; // valuetype SoftwareUnitValueType

  /**
  The Software Unit object class is a class of managed objects that provide
  administrable information associated with software (whether it be in the
  form of an executable file, such as program software, or a non-executable
  file, such as a data or cross-connect mapping table). The file type, file
  location, and file size are among the attributes identified in this object
  class. When the fileInformationPackage is present, the mandatory initial
  value of the dateOfCreation attribute is the time that the managed object
  is created. If backup operations are to be supported, then restore
  operations must be supported. The Information Backup and the Information
  Auto Backup packages will only exist in Software Unit managed objects that
  also have the Information Restore package or the Information Auto Restore
  package. The Information Restore and the Information Auto Restore packages
  will only exist in Software Unit managed objects that also have the
  Information Backup package or the Information Auto Backup package.
  When the attribute value change notification package (inherited from the
  superclass software) is present, the attributeValueChange notification
  defined in ITU-T Rec. X.780 shall be emitted when the value of one of
  the following attribute changes:
  - Affected Objects
  - Alarm Status
  - Applied Patches
  - Availability Status
  - Current Problem List
  - Future Auto Backup Destination
  - Future Auto Backup Trigger Threshold
  - Future Auto Restore Allowed
  - Future Auto Restore Source
  - Procedural Status
  - User Label
  - Version

  Because some of the above attributes are in conditional packages, the
  behaviour for emitting the attributeValueChange notification applies only
  when the corresponding conditional packages are present in the managed
  object.
  */

/**
```

9.6 SoftwareUnit interface

```
*/
  /**
  Software Unit managed object
  */

  interface SoftwareUnit : itut_m3120::Software
  {
    /**
    OperationalState and AdministrativeState from the Software managed object
    are now required with the Software Unit managed object. This means that
    the Packages attribute for Software Unit will always contain the
    "itut_m3120::administrativeOperationalStatesPackage" string
    */
```

```
/**
Availability Status is described in ITU-T Rec. X.731. Additional
behaviour found in 6.3.2
*/

AvailabilityStatusSetType availabilityStatusGet ()
   raises (itut_x780::ApplicationError);

/**
Procedural Status is described in ITU-T Rec. X.731. Additional
behaviour found in 6.3.2
*/

ProceduralStatusSetType proceduralStatusGet ()
   raises (itut_x780::ApplicationError);

/**
Applied Patches identifies the patches that have been applied to and
still exist in the Software Unit which is represented by the Software
Unit object instance. Patches are updates to software. The value of this
attribute is read-only and automatically updated when a patch is
performed on the software. The syntax of this attribute is a sequence of
patch identifiers,where a patch identifier is a choice of object instance
(if the patch is represented by a Software Unit managed object) or
graphic string (if the patch is not represented by a Software Unit
managed object).

Note that the optional Applied Patches attribute may need to be
internally maintained when it is not provided in the Managed Object
class. The Applied Patches must be maintained if the Install or Revert
services are supported (even if the Applied Patches attribute is not in
the Managed Object class).

PRESENT IF an instance supports software patching
*/

AppliedPatchesSeqType appliedPatchesGet ()
   raises (itut_x780::ApplicationError,
   NOappliedPatchPackage);

/**
Check Sum identifies the checksum of the software information represented
by the Software Unit object instance. PRESENT IF an instance supports
check sum validation
*/

CheckSumType checkSumGet ()
   raises (itut_x780::ApplicationError,
   NOcheckSumPackage);

/**
dateOfCreation indicates the time of creation of the managed object. The
syntax of this attribute is of GeneralizedTime type. PRESENT IF an
instance supports file information
*/

DateOfCreationType dateOfCreationGet ()
   raises (itut_x780::ApplicationError,
       NOfileInformationPackage);
```

```
/**
identityOfCreator identifies the entity that creates the managed object.
This may be an empty string if the creator identity is not known. PRESENT
IF an instance supports file information
*/

IdentityOfCreatorType identityOfCreatorGet ()
   raises (itut_x780::ApplicationError,
       NOfileInformationPackage);

/**
dateOfLastModification identifies the time of the last, or most recent
modification (e.g., patching, reverting, installing, delivering) to the
information represented by the Software Unit object instance. Valid
values for this attribute are GeneralizedTime or NULL if the information
has not been modified. PRESENT IF an instance supports file information
*/

DateOfLastModificationType dateOfLastModificationGet ()
   raises (itut_x780::ApplicationError,
       NOfileInformationPackage);

/**
identityOfLastModifier identifies the last, or most recent, modifier of
the information represented by the Software Unit object instance. This
may be an empty string if the last modifier identity is not known.
PRESENT IF an instance supports file information
*/

IdentityOfLastModifierType identityOfLastModifierGet ()
   raises (itut_x780::ApplicationError,
       NOfileInformationPackage);

/**
dateDelivered identifies the time that the information represented by the
Software Unit object instance was delivered to the managed system. Valid
values for this attribute are GeneralizedTime or NULL if the information
has not been delivered. PRESENT IF an instance supports file information
*/

DateDeliveredType dateDeliveredGet ()
   raises (itut_x780::ApplicationError,
       NOfileInformationPackage);

/**
dateInstalled identifies the time that the information represented by the
Software Unit object instance was installed. Valid values for this
attribute are GeneralizedTime or NULL if the information has not been
installed. PRESENT IF an instance supports file information
*/

DateInstalledType dateInstalledGet ()
   raises (itut_x780::ApplicationError,
       NOfileInformationPackage);

/**
fileLocation specifies the full address(es) (either logical or physical)
of the Software Unit object. The format of the address is implementation-
dependent, conforming to the file-addressing conventions of the
particular managed system in question. An empty set of this attribute
indicates that the information to which the Software Unit managed object
applies has not yet been installed in the managed system. PRESENT IF an
instance supports representation of a file
*/
```

```
FileLocationSetType fileLocationGet ()
   raises (itut_x780::ApplicationError,
       NOfilePackage);

/**
fileSize indicates the size of the Software Unit managed object. PRESENT
IF an instance supports representation of a file
*/

FileSizeType fileSizeGet ()
   raises (itut_x780::ApplicationError,
       NOfilePackage);

/**
fileType indicates the type of the Software Unit. Possible Software Unit
types are unstructured binary file (e.g., executable file), unstructured
text file (e.g., non-executable file), block special file, etc. PRESENT
IF an instance supports representation of a file
*/

FileTypeType fileTypeGet ()
   raises (itut_x780::ApplicationError,
       NOfilePackage);

/**
futureAutoBackupTriggerThreshold specifies the threshold that will
trigger an automatic backup for the information represented by the object
instance. The threshold is defined as the number of time that the
information has been modified. Once the information has been modified for
that number of times, an automatic backup will be performed. The backup
destination is specified in the attribute
futureAutomacticBackupDestination. Such backups are carried out in
addition to other scheduled periodic backup. At the completion of the
automatic backup, an backupReport notification shall be emitted from the
object. PRESENT IF an instance supports automatic backup
*/

FutureAutoBackupTriggerThresholdType
       futureAutoBackupTriggerThresholdGet ()
   raises (itut_x780::ApplicationError,
       NOinformationAutoBackupPackage);

void futureAutoBackupTriggerThresholdSet
   (in FutureAutoBackupTriggerThresholdType
       futureAutoBackupTriggerThreshold)
   raises (itut_x780::ApplicationError,
       NOinformationAutoBackupPackage);

/**
futureAutoBackupDestination specifies the destination to which the
information represented by this object instance will be backup. The
backup criteria is defined in the futureAutoBackupTriggerThreshold
attribute of the object instance. The destination can be another object
instance of the same object class exists in the same local managed system
or a remote open system (by using a particular file transfer protocol,
e.g., FTAM). PRESENT IF an instance supports automatic backup
*/

FutureAutoBackupDestinationType futureAutoBackupDestinationGet ()
   raises (itut_x780::ApplicationError,
       NOinformationAutoBackupPackage);
```

```
void futureAutoBackupDestinationSet
   (in FutureAutoBackupDestinationType futureAutoBackupDestination)
   raises (itut_x780::ApplicationError,
       NOinformationAutoBackupPackage);

/**
futureAutoRestoreSource specifies the source of the information to be
restored to the information represented by the managed object instance.
The source is either a local managed object or a remote system. The
criteria of triggering an automatic restore of information is system
specific. PRESENT IF an instance supports automatic restore
*/

FutureAutoRestoreSourceType futureAutoRestoreSourceGet ()
   raises (itut_x780::ApplicationError,
       NOinformationAutoRestorePackage);

void futureAutoRestoreSourceSet
   (in FutureAutoRestoreSourceType futureAutoRestoreSource)
   raises (itut_x780::ApplicationError,
       NOinformationAutoRestorePackage);

/**
futureAutoRestoreAllowed specifies whether automatic restore of the
information represented by this manage object instance is allowed. The
syntax of this attribute is of boolean type with the value TRUE meaning
allowed, and FALSE meaning not allowed. The criteria that triggers
automatic information restore is system specific. PRESENT IF an instance
supports automatic restore
*/

FutureAutoRestoreAllowedType futureAutoRestoreAllowedGet ()
   raises (itut_x780::ApplicationError,
       NOinformationAutoRestorePackage);

void futureAutoRestoreAllowedSet
   (in FutureAutoRestoreAllowedType futureAutoRestoreAllowed)
   raises (itut_x780::ApplicationError,
       NOinformationAutoRestorePackage);

/**
lastBackupTime identifies the time of the last backup on the information
represented by the managed object instance. Valid values for this
attribute are GeneralizedTime or NULL (if no backup has been performed on
the information). PRESENT IF an instance supports the Backup operation
*/

LastBackupTimeType lastBackupTimeGet ()
   raises (itut_x780::ApplicationError,
       NOinformationBackupPackage);

/**
lastBackupDestination identifies the destination, if exists, to which the
information represented by the managed object is backed up. PRESENT IF an
instance supports the Backup operation
*/

LastBackupDestinationType lastBackupDestinationGet ()
   raises (itut_x780::ApplicationError,
       NOinformationBackupPackage);

/**
The backup service is used by a managing system to request performing a
backup on the information represented by the target object instance
```

```
(i.e., managed object representing the software being backed up).
Following successful validation of the argument, the Backup operation
will immediately return. A Backup Report notification will be issued
following the completion of the target object backup.

@param backupDestination   This indicates the destination to which the
                           information will be backed up. Possible
                           destinations are:
                           - A local managed object - In this case, the
                           Backup operation will be performed internally
                           in the managed system. The information will be
                           backed up to the supplied managed object
                           instance.
                           - Off-line choice – In this case, the backup
                           information will be transferred off-line to the
                           remote system by some local means.
PRESENT IF an instance supports the Backup operation
*/

void backup
   (in BackupDestinationType backupDestionation)
   raises (itut_x780::ApplicationError,
       NOinformationBackupPackage,
       BackupSoftwareProcessingFailure,
       ConcurrentOperationRequestFailure);

/**
lastRestoreTime identifies the time of the last restore on the
information represented by the managed object instance. Valid values for
this attribute are GeneralizedTime or NULL (if no restore has been
performed on the information). PRESENT IF an instance supports the
Restore operation or automatic restore
*/

LastRestoreTimeType lastRestoreTimeGet ()
   raises (itut_x780::ApplicationError,
       NOinformationRestorePackage,
       NOinformationAutoRestorePackage);

/**
lastRestoreSource identifies the source, if exists, from which the
information represented by the managed object is restored. PRESENT IF an
instance supports the Restore operation or automatic restore
*/

LastRestoreSourceType lastRestoreSourceGet ()
   raises (itut_x780::ApplicationError,
       NOinformationRestorePackage,
       NOinformationAutoRestorePackage);

/**
The restore service is used by a managing system to request performing a
restore on the information represented by the target object instance.
Following successful validation of the argument, the Restore operation
will immediately return. A Restore Report notification will be issued
following the completion of the target object restore.

@param restoreSource       This indicates the source to which the
                           information will be restored. Possible sources
                           are:
                           - A local managed object of the same class as
                           the one this operation is applied to. In this
                           case, the Restore operation will be performed
                           internally in the managed system.
```

```
                          - Off-line choice – In this case, the restore
                          information will be transferred off-line from
                          the remote system by using a locally chosen
                          file transfer protocol.
PRESENT IF an instance supports the Restore operation
*/

void restore
   (in RestoreSourceType restoreSource)
   raises (itut_x780::ApplicationError,
       NOinformationRestorePackage,
       RestoreSoftwareProcessingFailure,
       ConcurrentOperationRequestFailure);

/**
The install service is used by a managing system to instruct a managed
system to install a delivered Software Unit object instance. If
applicable, the install service will update the value of the Applied
Patches attribute.

@param targetSoftware     This indicates the source of the software to be
                          installed. The source must be unique, from an
                          Applied Patches perspective. The source may be
                          one or more of the following:
                          - Patch Id – System-specific identifier.
                          - Patch Pointer - Software Unit (or Executable
                          Software) managed object class.
@return                   The Install service will automatically return
                          the value of the Applied Patches attribute of
                          the Software Unit object instance to which the
                          service is directed.
PRESENT IF an instance supports the install operation
*/

InstallReplyType install
   (in TargetSoftwareSetType targetSoftware)
   raises (itut_x780::ApplicationError,
       NOinstallPackage,
       InstallSoftwareProcessingFailure,
       OperationStateMismatch,
       ConcurrentOperationRequestFailure);

/**
noteField contains any information or comments associated with the
managed object, including any specific installation instructions, startup
parameters and values, information necessary to activate features of the
managed object, etc. PRESENT IF an instance supports it
*/

NoteFieldType noteFieldGet ()
   raises (itut_x780::ApplicationError,
       NOnoteFieldPackage);

void noteFieldSet
   (in NoteFieldType noteField)
   raises (itut_x780::ApplicationError,
       NOnoteFieldPackage);
```

```
/**
The revert service is used by a managing system (e.g., OS) to instruct a
managed system to revert an applied patch or set of patches of the
software represented by the Software Unit managed object. If the Revert
service successfully reverts all patches that have thus far been
installed (i.e., the Applied Patches attribute is empty), the Software
Unit internal state will be changed from Installed to Delivered.

@param revertInfo          The Revert Information parameter shall indicate
                           one or more previously applied patches to which
                           will be uninstalled. Each applied patch
                           identifier is a choice of a system-specific
                           identifier or a Software Unit object instance,
                           depending on the values originally supplied in
                           a previous Install operation on this Software
                           Unit object instance.
@return                    The Revert service will automatically return
                           the value of the Applied Patches attribute of
                           the Software Unit object instance to which the
                           service is directed.
PRESENT IF an instance supports it
*/

RevertReplyType revert
   (in RevertInfoSetType revertInfo)
   raises (itut_x780::ApplicationError,
       NOrevertPackage,
       OperationStateMismatch,
       RevertSoftwareProcessingFailure,
       ConcurrentOperationRequestFailure);

/**
Usage State is described in ITU-T Rec. X.731. PRESENT IF an instance
supports it
*/

UsageStateType usageStateGet ()
   raises (itut_x780::ApplicationError,
       NOusageStatePackage);

/**
The validate service is used by a managing system to request performing a
validation on the information represented by the Software Unit object
instance.

@param validateInfo        This indicates what type of validation should
                           be performed. Possible types are:
                           - Registered Validation – In this case, the
                           validation information is provided via another
                           specified managed object.
                           - Default Validation – In this case, the
                           default validation for this specific managed
                           object instance will be used.
PRESENT IF an instance supports it
*/

void validate
   (in ValidateInfoType validateInfo)
   raises (itut_x780::ApplicationError,
       NOvalidationPackage,
       OperationStateMismatch,
       ValidateSoftwareProcessingFailure,
       ConcurrentOperationRequestFailure);
```

```
      /**
      The Alarm Effect On Service Parameter in ITU-T Rec. X.744 has been
      incorporated into the processingErrorAlarm parameters.

      The Processing Error Alarm notification in the Software managed object
      class is now mandatory.
      */

      MANDATORY_NOTIFICATION(
         itut_x780::Notifications, processingErrorAlarm)

      /**
      The Backup Report notification is emitted to report a backup of the
      information represented by this object. This is initiated by the Backup
      operation. The backup destination may be local (i.e., backup to another
      Software Unit object within the local managed system) or off-line to a
      remote system by using a particular file transfer protocol (e.g., FTAM).
      The result of the backup, i.e., success or failure, will be reported in
      this notification. PRESENT IF an instance supports the Backup operation
      or automatic backup
      */

      CONDITIONAL_NOTIFICATION(
         itut_x744d1::Notifications, backupReport, informationBackupPackage)

      CONDITIONAL_NOTIFICATION(
         itut_x744d1::Notifications, backupReport,
         informationAutoBackupPackage)

      /**
      The Restore Report notification is emitted to report a restore of a
      managed object from a previous backup. The restore may have been
      initiated automatically (according to the Future Auto Restore Source Type
      and Future Auto Restore Allowed Type attributes and system specific
      criteria), through management request (via the Restore operation) or
      initiated by the managed system. The Restore Source may be local (i.e.,
      restore from another Software Unit object within the local managed
      system) or off-line from a remote system by using a particular file
      transfer protocol (e.g., FTAM). The result of the restore will be
      reported in this notification. PRESENT IF an instance supports the
      Restore operation or automatic restore
      */

      CONDITIONAL_NOTIFICATION(
         itut_x744d1::Notifications, restoreReport,
         informationRestorePackage)

      CONDITIONAL_NOTIFICATION(
         itut_x744d1::Notifications, restoreReport,
         informationAutoRestorePackage)

      /**
      The Validate Report notification is emitted to report a validation of a
      managed object. The result of the validate operation will be reported in
      this notification. PRESENT IF an instance supports the Validate operation
      */

      CONDITIONAL_NOTIFICATION(
         itut_x744d1::Notifications, validateReport, validationPackage)

    }; // interface SoftwareUnit

/**
```

9.7 SoftwareUnit_F interface

```
*/
     /**
     Software Unit Facade managed object - see ITU-T Rec. X.780.1
     */

     interface SoftwareUnit_F : itut_m3120::Software_F
     {
       /**
       OperationalState and AdministrativeState from the Software managed object
       are now required with the Software Unit managed object. This means that
       the Packages attribute for Software Unit will always contain the
       "itut_m3120::administrativeOperationalStatesPackage" string
       */

       /**
       Availability Status is described in ITU-T Rec. X.731. Additional
       behaviour found in 6.3.2
       */

       AvailabilityStatusSetType availabilityStatusGet
          (in MONameType name)
          raises (itut_x780::ApplicationError);

       /**
       Procedural Status is described in ITU-T Rec. X.731. Additional
       behaviour found in 6.3.2
       */

       ProceduralStatusSetType proceduralStatusGet
          (in MONameType name)
          raises (itut_x780::ApplicationError);

       /**
       Applied Patches identifies the patches that have been applied to and
       still exist in the Software Unit which is represented by the Software
       Unit object instance. Patches are updates to software. The value of this
       attribute is read-only and automatically updated when a patch is
       performed on the software. The syntax of this attribute is a sequence of
       patch identifiers,where a patch identifier is a choice of object instance
       (if the patch is represented by a Software Unit managed object) or
       graphic string (if the patch is not represented by a Software Unit
       managed object).

       Note that the optional Applied Patches attribute may need to be
       internally maintained when it is not provided in the Managed Object
       class. The Applied Patches must be maintained if the Install or Revert
       services are supported (even if the Applied Patches attribute is not in
       the Managed Object class).

       PRESENT IF an instance supports software patching
       */

       AppliedPatchesSeqType appliedPatchesGet
          (in MONameType name)
          raises (itut_x780::ApplicationError,
              NOappliedPatchPackage);
```

```
/**
Check Sum identifies the checksum of the software information represented
by the Software Unit object instance. PRESENT IF an instance supports
check sum validation
*/

CheckSumType checkSumGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
   NOcheckSumPackage);

/**
dateOfCreation indicates the time of creation of the managed object. The
syntax of this attribute is of GeneralizedTime type. PRESENT IF an
instance supports file information
*/

DateOfCreationType dateOfCreationGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOfileInformationPackage);

/**
identityOfCreator identifies the entity that creates the managed object.
This may be an empty string if the creator identity is not known. PRESENT
IF an instance supports file information
*/

IdentityOfCreatorType identityOfCreatorGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOfileInformationPackage);

/**
dateOfLastModification identifies the time of the last, or most recent
modification (e.g., patching, reverting, installing, delivering) to the
information represented by the Software Unit object instance. Valid
values for this attribute are GeneralizedTime or NULL if the information
has not been modified. PRESENT IF an instance supports file information
*/

DateOfLastModificationType dateOfLastModificationGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOfileInformationPackage);

/**
identityOfLastModifier identifies the last, or most recent, modifier of
the information represented by the Software Unit object instance. This
may be an empty string if the last modifier identity is not known.
PRESENT IF an instance supports file information
*/

IdentityOfLastModifierType identityOfLastModifierGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOfileInformationPackage);
```

```
/**
dateDelivered identifies the time that the information represented by the
Software Unit object instance was delivered to the managed system. Valid
values for this attribute are GeneralizedTime or NULL if the information
has not been delivered. PRESENT IF an instance supports file information
*/

DateDeliveredType dateDeliveredGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOfileInformationPackage);

/**
dateInstalled identifies the time that the information represented by the
Software Unit object instance was installed. Valid values for this
attribute are GeneralizedTime or NULL if the information has not been
installed. PRESENT IF an instance supports file information
*/

DateInstalledType dateInstalledGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOfileInformationPackage);

/**
fileLocation specifies the full address(es) (either logical or physical)
of the Software Unit object. The format of the address is implementation-
dependent, conforming to the file-addressing conventions of the
particular managed system in question. An empty set of this attribute
indicates that the information to which the Software Unit managed object
applies has not yet been installed in the managed system. PRESENT IF an
instance supports representation of a file
*/

FileLocationSetType fileLocationGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOfilePackage);

/**
fileSize indicates the size of the Software Unit managed object. PRESENT
IF an instance supports representation of a file.
*/

FileSizeType fileSizeGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOfilePackage);

/**
fileType indicates the type of the Software Unit. Possible Software Unit
types are unstructured binary file (e.g., executable file), unstructured
text file (e.g., non-executable file), block special file, etc. PRESENT
IF an instance supports representation of a file
*/

FileTypeType fileTypeGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOfilePackage);
```

```
/**
futureAutoBackupTriggerThreshold specifies the threshold that will
trigger an automatic backup for the information represented by the object
instance. The threshold is defined as the number of time that the
information has been modified. Once the information has been modified for
that number of times, an automatic backup will be performed. The backup
destination is specified in the attribute
futureAutomacticBackupDestination. Such backups are carried out in
addition to other scheduled periodic backup. At the completion of the
automatic backup, an backupReport notification shall be emitted from the
object. PRESENT IF an instance supports automatic backup
*/

FutureAutoBackupTriggerThresholdType
       FutureAutoBackupTriggerThresholdGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOinformationAutoBackupPackage);

void futureAutoBackupTriggerThresholdSet
   (in MONameType name,
   in FutureAutoBackupTriggerThresholdType
       futureAutoBackupTriggerThreshold)
   raises (itut_x780::ApplicationError,
       NOinformationAutoBackupPackage);

/**
futureAutoBackupDestination specifies the destination to which the
information represented by this object instance will be backup. The
backup criteria is defined in the futureAutoBackupTriggerThreshold
attribute of the object instance. The destination can be another object
instance of the same object class exists in the same local managed system
or a remote open system (by using a particular file transfer protocol,
e.g., FTAM). PRESENT IF an instance supports automatic backup
*/

FutureAutoBackupDestinationType futureAutoBackupDestinationGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOinformationAutoBackupPackage);

void futureAutoBackupDestinationSet
   (in MONameType name,
   in FutureAutoBackupDestinationType futureAutoBackupDestination)
   raises (itut_x780::ApplicationError,
       NOinformationAutoBackupPackage);

/**
futureAutoRestoreSource specifies the source of the information to be
restored to the information represented by the managed object instance.
The source is either a local managed object or a remote system. The
criteria of triggering an automatic restore of information is system
specific. PRESENT IF an instance supports automatic restore
*/

FutureAutoRestoreSourceType futureAutoRestoreSourceGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOinformationAutoRestorePackage);
```

```
void futureAutoRestoreSourceSet
   (in MONameType name,
   in FutureAutoRestoreSourceType futureAutoRestoreSource)
   raises (itut_x780::ApplicationError,
       NOinformationAutoRestorePackage);

/**
futureAutoRestoreAllowed specifies whether automatic restore of the
information represented by this manage object instance is allowed. The
syntax of this attribute is of boolean type with the value TRUE meaning
allowed, and FALSE meaning not allowed. The criteria that triggers
automatic information restore is system specific. PRESENT IF an instance
supports automatic restore
*/

FutureAutoRestoreAllowedType futureAutoRestoreAllowedGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOinformationAutoRestorePackage);

void futureAutoRestoreAllowedSet
   (in MONameType name,
   in FutureAutoRestoreAllowedType futureAutoRestoreAllowed)
   raises (itut_x780::ApplicationError,
       NOinformationAutoRestorePackage);

/**
lastBackupTime identifies the time of the last backup on the information
represented by the managed object instance. Valid values for this
attribute are GeneralizedTime or NULL (if no backup has been performed on
the information). PRESENT IF an instance supports the Backup operation
*/

LastBackupTimeType lastBackupTimeGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOinformationBackupPackage);

/**
lastBackupDestination identifies the destination, if exists, to which the
information represented by the managed object is backed up. PRESENT IF an
instance supports the Backup operation
*/

LastBackupDestinationType lastBackupDestinationGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOinformationBackupPackage);

/**
The backup service is used by a managing system to request performing a
backup on the information represented by the target object instance
(i.e., managed object representing the software being backed up).
Following successful validation of the argument, the Backup operation
will immediately return. A Backup Report notification will be issued
following the completion of the target object backup.

@param name                Software Unit managed object instance name
@param backupDestination   This indicates the destination to which the
                           information will be backed up. Possible
                           destinations are:
                           - A local managed object - In this case, the
                           Backup operation will be performed internally
                           in the managed system. The information will be
```

```
                          backed up to the supplied managed object
                          instance.
                          - Off-line choice – In this case, the backup
                          information will be transferred off-line to the
                          remote system by some local means.
PRESENT IF an instance supports the Backup operation
*/

void backup
   (in MONameType name,
   in BackupDestinationType backupDestionation)
   raises (itut_x780::ApplicationError,
       NOinformationBackupPackage,
       BackupSoftwareProcessingFailure,
       ConcurrentOperationRequestFailure);

/**
lastRestoreTime identifies the time of the last restore on the
information represented by the managed object instance. Valid values for
this attribute are GeneralizedTime or NULL (if no restore has been
performed on the information). PRESENT IF an instance supports the
Restore operation or automatic restore
*/

LastRestoreTimeType lastRestoreTimeGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOinformationRestorePackage,
       NOinformationAutoRestorePackage);

/**
lastRestoreSource identifies the source, if exists, from which the
information represented by the managed object is restored. PRESENT IF an
instance supports the Restore operation or automatic restore
*/

LastRestoreSourceType lastRestoreSourceGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOinformationRestorePackage,
       NOinformationAutoRestorePackage);

/**
The restore service is used by a managing system to request performing a
restore on the information represented by the target object instance.
Following successful validation of the argument, the Restore operation
will immediately return. A Restore Report notification will be issued
following the completion of the target object restore.

@param name               Software Unit managed object instance name
@param restoreSource      This indicates the source to which the
                          information will be restored. Possible sources
                          are:
                          - A local managed object of the same class as
                          the one this operation is applied to. In this
                          case, the Restore operation will be performed
                          internally in the managed system.
                          - Off-line choice – In this case, the restore
                          information will be transferred off-line from
                          the remote system by using a locally chosen
                          file transfer protocol.
PRESENT IF an instance supports the Restore operation
*/
```

```
void restore
   (in MONameType name,
   in RestoreSourceType restoreSource)
   raises (itut_x780::ApplicationError,
       NOinformationRestorePackage,
       RestoreSoftwareProcessingFailure,
       ConcurrentOperationRequestFailure);

/**
The install service is used by a managing system to instruct a managed
system to install a delivered Software Unit object instance. If
applicable, the install service will update the value of the Applied
Patches attribute.

@param name                Software Unit managed object instance name
@param targetSoftware      This indicates the source of the software to be
                           installed. The source must be unique, from an
                           Applied Patches perspective. The source may be
                           one or more of the following:
                           - Patch Id – System-specific identifier.
                           - Patch Pointer - Software Unit (or Executable
                           Software) managed object class.
@return                    The Install service will automatically return
                           the value of the Applied Patches attribute of
                           the Software Unit object instance to which the
                           service is directed.
PRESENT IF an instance supports the install operation
*/

InstallReplyType install
   (in MONameType name,
   in TargetSoftwareSetType targetSoftware)
   raises (itut_x780::ApplicationError,
       NOinstallPackage,
       InstallSoftwareProcessingFailure,
       OperationStateMismatch,
       ConcurrentOperationRequestFailure);

/**
noteField contains any information or comments associated with the
object, including any specific installation instructions, startup
managed parameters and values, information necessary to activate features
of the managed object, etc. PRESENT IF an instance supports it
*/

NoteFieldType noteFieldGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOnoteFieldPackage);

void noteFieldSet
   (in MONameType name,
   in NoteFieldType noteField)
   raises (itut_x780::ApplicationError,
       NOnoteFieldPackage);

/**
The revert service is used by a managing system (e.g., OS) to instruct a
managed system to revert an applied patch or set of patches of the
software represented by the Software Unit managed object. If the Revert
service successfully reverts all patches that have thus far been
installed (i.e., the Applied Patches attribute is empty), the Software
Unit internal state will be changed from Installed to Delivered.
```

```
@param name                Software Unit managed object instance name
@param revertInfo          The Revert Information parameter shall indicate
                           one or more previously applied patches to which
                           will be uninstalled. Each applied patch
                           identifier is a choice of a system-specific
                           identifier or a Software Unit object instance,
                           depending on the values originally supplied in
                           a previous Install operation on this Software
                           Unit object instance.
@return                    The Revert service will automatically return
                           the value of the Applied Patches attribute of
                           the Software Unit object instance to which the
                           service is directed.
PRESENT IF an instance supports it
*/

RevertReplyType revert
   (in MONameType name,
   in RevertInfoSetType revertInfo)
   raises (itut_x780::ApplicationError,
       NOrevertPackage,
       OperationStateMismatch,
       RevertSoftwareProcessingFailure,
       ConcurrentOperationRequestFailure);

/**
Usage State is described in ITU-T Rec. X.731. PRESENT IF an instance
supports it
*/

UsageStateType usageStateGet
   (in MONameType name)
   raises (itut_x780::ApplicationError,
       NOusageStatePackage);

/**
The validate service is used by a managing system to request performing a
validation on the information represented by the Software Unit object
instance.

@param name                Software Unit managed object instance name
@param validateInfo        This indicates what type of validation should
                           be performed. Possible types are:
                           - Registered Validation – In this case, the
                           validation information is provided via another
                           specified managed object.
                           - Default Validation – In this case, the
                           default validation for this specific managed
                           object instance will be used.
PRESENT IF an instance supports it
*/

void validate
   (in MONameType name,
   in ValidateInfoType validateInfo)
   raises (itut_x780::ApplicationError,
       NOvalidationPackage,
       OperationStateMismatch,
       ValidateSoftwareProcessingFailure,
       ConcurrentOperationRequestFailure);
```

```
      /**
      The Alarm Effect On Service Parameter in ITU-T Rec. X.744 has been
      incorporated into the processingErrorAlarm parameters.

      The Processing Error Alarm notification in the Software managed object
      class is now mandatory.
      */

      MANDATORY_NOTIFICATION(
         itut_x780::Notifications, processingErrorAlarm)

      /**
      The Backup Report notification is emitted to report a backup of the
      information represented by this object. This is initiated by the Backup
      operation. The backup destination may be local (i.e., backup to another
      Software Unit object within the local managed system) or off-line to a
      remote system by using a particular file transfer protocol (e.g., FTAM).
      The result of the backup, i.e., success or failure, will be reported in
      this notification. PRESENT IF an instance supports the Backup operation
      or automatic backup
      */

      CONDITIONAL_NOTIFICATION(
         itut_x744d1::Notifications, backupReport, informationBackupPackage)

      CONDITIONAL_NOTIFICATION(
         itut_x744d1::Notifications, backupReport,
         informationAutoBackupPackage)

      /**
      The Restore Report notification is emitted to report a restore of a
      managed object from a previous backup. The restore may have been
      initiated automatically (according to the Future Auto Restore Source Type
      and Future Auto Restore Allowed Type attributes and system specific
      criteria), through management request (via the Restore operation) or
      initiated by the managed system. The Restore Source may be local (i.e.,
      restore from another Software Unit object within the local managed
      system) or off-line from a remote system by using a particular file
      transfer protocol (e.g., FTAM). The result of the restore will be
      reported in this notification. PRESENT IF an instance supports the
      Restore operation or automatic restore
      */

      CONDITIONAL_NOTIFICATION(
         itut_x744d1::Notifications, restoreReport,
         informationRestorePackage)

      CONDITIONAL_NOTIFICATION(
         itut_x744d1::Notifications, restoreReport,
         informationAutoRestorePackage)

      /**
      The Validate Report notification is emitted to report a validation of a
      managed object. The result of the validate operation will be reported in
      this notification. PRESENT IF an instance supports the Validate operation
      */

      CONDITIONAL_NOTIFICATION(
         itut_x744d1::Notifications, validateReport, validationPackage)

   }; // interface SoftwareUnit_F

/**
```

9.8 SoftwareUnitFactory interface

```
*/
     /**
     Creation and Deletion for Software Unit
     */

     interface SoftwareUnitFactory : itut_x780::ManagedObjectFactory
     {
       itut_x780::ManagedObject create
          (in NameBindingType nameBinding,
          in MONameType superior,
          in string reqID,   // auto naming if empty string
          out MONameType name,
          in StringSetType packageNameList,
          in AdministrativeStateType adminstrativeState,
              // conditional
              // itut_m3120::administrativeOperationalStatePackage
              // GET-REPLACE
          in AlarmSeverityAssignmentProfileNameType profile,
              // conditional
              // itut_m3120::alarmSeverityAssignmentPointerPackage
              // GET-REPLACE
          in Istring userLabel,
              // conditional
              // itut_m3120::userLabelPackage
              // GET-REPLACE
          in Istring vendorName,
              // conditional
              // itut_m3120::vendorNamePackage
              // GET-REPLACE
          in Istring version,
              // conditional
              // itut_m3120::versionPackage
              // GET-REPLACE
          in ArcProbableCauseSetType arcProbableCauseList,
              // conditional
              // itut_m3120::arcPackage
              // GET-REPLACE, ADD-REMOVE
          in ArcIntervalProfileNameType arcIntervalProfilePointer,
              // conditional
              // itut_m3120::arcPackage
              // GET-REPLACE
          in ArcTimeType arcManagementRequestedInterval,
              // conditional
              // itut_m3120::arcPackage
              // GET-REPLACE
          in AvailabilityStatusSetType availabilityStatus,
              // GET
              // SET-BY-CREATE
          in FutureAutoBackupTriggerThresholdType
                       futureAutoBackupTriggerThreshold,
              // GET-REPLACE
              // informationAutoBackupPackage
          in FutureAutoBackupDestinationType futureAutoBackupDestination,
              // GET-REPLACE
              // informationAutoBackupPackage
          in FutureAutoRestoreSourceType futureAutoRestoreSource,
              // GET-REPLACE
              // informationAutoRestorePackage
          in FutureAutoRestoreAllowedType futureAutoRestoreAllowed,
              // GET-REPLACE
              // informationAutoRestorePackage
```

```
        in NoteFieldType noteField
            // GET-REPLACE
            // noteFieldPackage
        )
        raises (itut_x780::ApplicationError,
            itut_x780::CreateError);

    }; // interface SoftwareUnitFactory

/**
```

9.9 Executable software

```
*/
    /**
    This valuetype is used to retrieve all attributes
    */

    valuetype ExecutableSoftwareValueType : truncatable SoftwareUnitValueType
    {
    }; // valuetype ExecutableSoftwareValueType

    /**
    The executableSoftware object class is a class of managed objects that
    provide administrable information associated with an executable program in
    the managed system. The actual executable program (that may consist of code
    segments with or without data segments, etc.) may be in a non-standard,
    machine-dependent format that is generally unreadable by managing system and
    the rest of the outside world. An operation called executeProgram
    (conditionally) can be used to execute the program represented by the
    executableSoftware object instance. The usageState attribute is used to
    indicate if there are any active executions of the program.
    */

/**
```

9.10 ExecutableSoftware interface

```
*/
    /**
    Executable Software managed object
    */

    interface ExecutableSoftware : SoftwareUnit
    {
      /**
      UsageState from the Software Unit managed object is now required with the
      Executable Software managed object. This means that the Packages
      attribute for Executable Software will always contain the
      "itut_x744d1::usageStatePackage" string
      */

      /**
      The Execute Program service is used by a managing system to initiate the
      execution of a program represented by an Executable Software object. To
      execute the Execute Program operation, the Executable Software must be in
      the Installed internal state, the Administrative State must be Unlocked,
      the Operational State must be Enabled and the Usage State must either be
      Active or Idle.

      @param additionalInfo      This defines the parameters used in the
                                 execution of the software.
      @return                    A successful request will be confirmed with
```

```
                              information including the Process Id, Process
                              Owner, Initial start time and the supplied
                              Additional Information parameters.
      PRESENT IF an instance supports it
      */

      ExecuteProgramReplyType executeProgram
         (in AdditionalInformationSetType additionalInfo)
         raises (itut_x780::ApplicationError,
             NOexecuteProgramPackage,
             OperationStateMismatch,
             ExecuteProgramSoftwareProcessingFailure);

   }; // interface ExecutableSoftware

/**
```

9.11 ExecutableSoftware_F interface

```
*/
   /**
   Executable Unit Facade managed object - see ITU-T Rec. X.780.1
   */

   interface ExecutableSoftware_F : SoftwareUnit_F
   {
      /**
      UsageState from the Software Unit managed object is now required with the
      Executable Software managed object. This means that the Packages
      attribute for Executable Software will always contain the
      "itut_x744d1::usageStatePackage" string
      */

      /**
      The Execute Program service is used by a managing system to initiate the
      execution of a program represented by an Executable Software object. To
      execute the Execute Program operation, the Executable Software must be in
      the Installed internal state, the Administrative State must be Unlocked,
      the Operational State must be Enabled and the Usage State must either be
      Active or Idle.

      @param name             Executable Software managed object instance
                              name
      @param additionalInfo   This defines the parameters used in the
                              execution of the software.
      @return                 A successful request will be confirmed with
                              information including the Process Id, Process
                              Owner, Initial start time and the supplied
                              Additional Information parameters.
      PRESENT IF an instance supports it
      */

      ExecuteProgramReplyType executeProgram
         (in MONameType name,
         in AdditionalInformationSetType additionalInfo)
         raises (itut_x780::ApplicationError,
             NOexecuteProgramPackage,
             OperationStateMismatch,
             ExecuteProgramSoftwareProcessingFailure);

   }; // interface ExecutableSoftware_F

/**
```

9.12 ExecutableSoftwareFactory interface

```
*/
     /**
     Creation and Deletion for Executable Unit
     */

     interface ExecutableSoftwareFactory : itut_x780::ManagedObjectFactory
     {
       itut_x780::ManagedObject create
          (in NameBindingType nameBinding,
          in MONameType superior,
          in string reqID,   // auto naming if empty string
          out MONameType name,
          in StringSetType packageNameList,
          in AdministrativeStateType adminstrativeState,
              // conditional
              // itut_m3120::administrativeOperationalStatePackage
              // GET-REPLACE
          in AlarmSeverityAssignmentProfileNameType profile,
              // conditional
              // itut_m3120::alarmSeverityAssignmentPointerPackage
              // GET-REPLACE
          in Istring userLabel,
              // conditional
              // itut_m3120::userLabelPackage
              // GET-REPLACE
          in Istring vendorName,
              // conditional
              // itut_m3120::vendorNamePackage
              // GET-REPLACE
          in Istring version,
              // conditional
              // itut_m3120::versionPackage
              // GET-REPLACE
          in ArcProbableCauseSetType arcProbableCauseList,
              // conditional
              // itut_m3120::arcPackage
              // GET-REPLACE, ADD-REMOVE
          in ArcIntervalProfileNameType arcIntervalProfilePointer,
              // conditional
              // itut_m3120::arcPackage
              // GET-REPLACE
          in ArcTimeType arcManagementRequestedInterval,
              // conditional
              // itut_m3120::arcPackage
              // GET-REPLACE
          in AvailabilityStatusSetType availabilityStatus,
              // GET
              // SET-BY-CREATE
          in FutureAutoBackupTriggerThresholdType
              futureAutoBackupTriggerThreshold,
              // GET-REPLACE
              // informationAutoBackupPackage
          in FutureAutoBackupDestinationType futureAutoBackupDestination,
              // GET-REPLACE
              // informationAutoBackupPackage
          in FutureAutoRestoreSourceType futureAutoRestoreSource,
              // GET-REPLACE
              // informationAutoRestorePackage
          in FutureAutoRestoreAllowedType futureAutoRestoreAllowed,
              // GET-REPLACE
              // informationAutoRestorePackage
```

```
        in NoteFieldType noteField
            // GET-REPLACE
            // noteFieldPackage
        )
        raises (itut_x780::ApplicationError,
            itut_x780::CreateError);

    }; // interface ExecutableSoftwareFactory
```

/**

9.13 Software distributor

```
*/
    /**
    This valuetype is used to retrieve all attributes
    */

    valuetype SoftwareDistributorValueType :
        truncatable itut_x780::ManagedObjectValueType
    {
      public AdministrativeStateType administrativeState;
         // GET-REPLACE
      public OperationalStateType operationalState;
         // GET

    }; // valuetype SoftwareDistributorValueType

    /**
    A Software distributor managed object is a managed object which distributes
    software to the target managed system when it receives a Deliver operation
    from the managing system. This managed object notifies the result of the
    distribution when the distribution is completed. The
    stateChangeNotification defined in ITU-T Rec. X.780 shall be emitted if
    the value of the Administrative State or Operational State changes.
    */
```

/**

9.14 SoftwareDistributor interface

```
*/
    /**
    Software Distributor managed object
    */

    interface SoftwareDistributor : itut_x780::ManagedObject
    {
      /**
      Administrative State is described in ITU-T Rec. X.731
      */

      AdministrativeStateType administrativeStateGet ()
         raises (itut_x780::ApplicationError);

      void administrativeStateSet
         (in AdministrativeStateType administrativeState)
         raises (itut_x780::ApplicationError);

      /**
      Operational State is described in ITU-T Rec. X.731
      */
      OperationalStateType operationalStateGet ()
         raises (itut_x780::ApplicationError);

      /**
      The deliver service is used by a managing system to request distribution
      of software or a set of software. The Deliver operation information
      identifies the software that is to be distributed. The result of a
      Deliver operation is that a copy of the target software items is
      delivered to the target system in the Delivered internal state.

      Packaging of the software and the choice of transfer mechanism is a local
      matter and outside the scope of this ITU-T Rec.. For example, this
      information may be pre-configured or specified in the Deliver operation
      along with any other associated information.

      The result of successful completion is that the software to be
      distributed is copied to the target system; this may have the effect of
      Software Unit and/or Executable Software objects being created. After
      command completion, a Deliver Result notification is emitted.

      @param deliverId           This optional parameter indicates a unique
                                 identifier for this Deliver operation.
      @param targetSoftware      This indicates the source of the software to be
                                 delivered.
      @param targetSystem        This indicates the optional target destination
                                 for the software to be delivered. If this does
                                 not indicate a target destination, the system
                                 uses local means to determine the target
                                 destination.
      @param transferInfo        Application-specific transfer mechanism.
      @param additionalInfo      Additional application-specific information.
      */

      void deliver
         (in DeliverIdTypeOpt deliverId,
         in TargetSoftwareSetType targetSoftware,
         in DestinationType targetSystem,
         in TransferInfoType transferInfo,
         in AdditionalInformationSetType additionalInfo)
         raises (itut_x780::ApplicationError,
             OperationStateMismatch,
             DeliverSoftwareProcessingFailure);

      /**
      The Deliver Result notification is emitted from the managed object when
      the Deliver operation is completed. It contains the final results of the
      operation and may indicate either a pass or fail condition.
      */

      MANDATORY_NOTIFICATION(
         itut_x744d1::Notifications, deliverResultNotification)
      MANDATORY_NOTIFICATION(
         itut_x780::Notifications, objectCreation)
      MANDATORY_NOTIFICATION(
         itut_x780::Notifications, objectDeletion)
      MANDATORY_NOTIFICATION(
         itut_x780::Notifications, stateChange)

    }; // interface SoftwareDistributor

/**
```

9.15 SoftwareDistributor_F interface

```
*/
     /**
     Software Distributor Facade managed object – see ITU-T Rec. X.780.1
     */

     interface SoftwareDistributor_F : itut_x780::ManagedObject_F
     {
       /**
       Administrative State is described in ITU-T Rec. X.731
       */

       AdministrativeStateType administrativeStateGet
          (in MONameType name)
          raises (itut_x780::ApplicationError);

       void administrativeStateSet
          (in MONameType name,
          in AdministrativeStateType administrativeState)
          raises (itut_x780::ApplicationError);

       /**
       Operational State is described in ITU-T Rec. X.731
       */

       OperationalStateType operationalStateGet
          (in MONameType name)
          raises (itut_x780::ApplicationError);

       /**
       The deliver service is used by a managing system to request distribution
       of software or a set of software. The Deliver operation information
       identifies the software that is to be distributed. The result of a
       Deliver operation is that a copy of the target software items is
       delivered to the target system in the Delivered internal state.

       Packaging of the software and the choice of transfer mechanism is a local
       matter and outside the scope of this Recommendation. For example, this
       information may be pre-configured or specified in the Deliver operation
       along with any other associated information.

       The result of successful completion is that the software to be
       distributed is copied to the target system; this may have the effect of
       Software Unit and/or Executable Software objects being created. After
       command completion, a Deliver Result notification is emitted.

       @param name              Software Distributor managed object instance
                                name
       @param deliverId         This optional parameter indicates a unique
                                identifier for this Deliver operation.
       @param targetSoftware    This indicates the source of the software to be
                                delivered.
       @param targetSystem      This indicates the optional target destination
                                for the software to be delivered. If this does
                                not indicate a target destination, the system
                                uses local means to determine the target
                                destination.
       @param transferInfo      Application-specific transfer mechanism.
       @param additionalInfo    Additional application-specific information.
```

```
      */

      void deliver
         (in MONameType name,
         in DeliverIdTypeOpt deliverId,
         in TargetSoftwareSetType targetSoftware,
         in DestinationType targetSystem,
         in TransferInfoType transferInfo,
         in AdditionalInformationSetType additionalInfo)
         raises (itut_x780::ApplicationError,
             OperationStateMismatch,
             DeliverSoftwareProcessingFailure);

      /**
      The Deliver Result notification is emitted from the managed object when
      the Deliver operation is completed. It contains the final results of the
      operation and may indicate either a pass or fail condition.
      */

      MANDATORY_NOTIFICATION(
         itut_x744d1::Notifications, deliverResultNotification)
      MANDATORY_NOTIFICATION(
         itut_x780::Notifications, objectCreation)
      MANDATORY_NOTIFICATION(
         itut_x780::Notifications, objectDeletion)
      MANDATORY_NOTIFICATION(
         itut_x780::Notifications, stateChange)

    }; // interface SoftwareDistributor_F

/**
```

9.16 SoftwareDistributorFactory interface

```
*/
    /**
    Creation and Deletion for Software Distributor
    */

    interface SoftwareDistributorFactory : itut_x780::ManagedObjectFactory
    {
      itut_x780::ManagedObject create
         (in NameBindingType nameBinding,
         in MONameType superior,
         in string reqID,   // auto naming if empty string
         out MONameType name,
         in StringSetType packageNameList,
         in AdministrativeStateType administrativeState
              // GET-REPLACE
         )
         raises (itut_x780::ApplicationError,
              itut_x780::CreateError);

    }; // interface SoftwareDistributorFactory

/**
```

9.17 Notifications

```
*/

     interface Notifications
     {
       /**
       The Backup Report notification is emitted to report a backup of the
       information represented by this object.

       @param eventTime           Managed system's current time.
       @param source              Object emitting notification.
       @param request             Indicates who issued the request, either
                                  automatic, management or managed system
       @param backupResult        This will be one of the following, depending on
                                  success or failure of the backup and the value
                                  of the Backup operation parameters:
                                  - Failure Choice – Application error
                                  indication.
                                  - Off-Line Choice – The off-line backup
                                  location.
                                  - Local Object Choice – The backup object
                                  instance.
       */

       void backupReport (
          in ExternalTimeType eventTime,
          in MONameType source,
          in RequestType request,
          in BackupResultType backupResult
       );

       /**
       The Deliver Result notification is emitted from the managed object when
       the Deliver operation is completed.

       @param eventTime           Managed system's current time.
       @param source              Object emitting notification.
       @param deliverId           The unique Deliver Id supplied with the Deliver
                                  operation (when provided).
       @param deliver             The results of the Deliver operation. This is
                                  of type UIDType, so the results can be
                                  localized. Deliver results supplied in this
                                  Recommendation can be found in the
                                  DeliverResultConst module.
       @param additionalInfo      Application-specific information.
       */

       void deliverResultNotification (
          in ExternalTimeType eventTime,
          in MONameType source,
          in DeliverIdTypeOpt deliverId,
          in DeliverResultType deliver,
          in AdditionalInformationSetType additionalInfo
       );

       /**
       The Restore Report notification is emitted to report a restore of a
       managed object from a previous backup.

       @param eventTime           Managed system's current time.
       @param source              Object emitting notification.
       @param request             Indicates who issued the request, either
                                  automatic, management or managed system
```

```
  @param restoreResult      This will be one of the following, depending on
                            success or failure of the restore and the value
                            of the Future Auto Restore Source attribute:
                            - Failure Choice – Application error
                            indication.
                            - Off-Line Choice – The off-line restore
                            location.
                            - Local Object Choice – The restore object
                            instance.
  */

  void restoreReport (
     in ExternalTimeType eventTime,
     in MONameType source,
     in RequestType request,
     in RestoreResultType restoreResult
  );

  /**
  The Validate Report notification is emitted to report a validation of a
  managed object.

  @param eventTime          Managed system’s current time.
  @param source             Object emitting notification.
  @param validateInfo       Indicates the argument given to the validate
                            operation
  @param validateResult     This will be one of the following, depending on
                            success or failure of the validate operation:
                            - Pass Validation (with result)
                            - Fail Validation (with result)
                            - Pass Validation
                            - Fail Validation (with error)
  */

  void validateReport (
     in ExternalTimeType eventTime,
     in MONameType source,
     in ValidateInfoType validateInfo,
     in ValidateResultType validateResult
  );
};

/**
Notification constants
*/

const string backupReportTypeName =
  "itut_x744d1::Notifications::backupReport";
const string deliverResultNotificationTypeName =
  "itut_x744d1::Notifications::deliverResultNotification";
const string restoreReportTypeName =
  "itut_x744d1::Notifications::restoreReport";
const string validateReportTypeName =
  "itut_x744d1::Notifications::validateReport";
const string additionalInfoName = "additionalInfo";
const string backupResultName = "backupResult";
const string deliverIdName = "deliverId";
const string deliverName = "deliver";
const string eventTimeName = "eventTime";
const string sourceName = "source";
```

```
     const string restoreResultName = "restoreResult";
     const string requestName = "request";
     const string validateInfoName = "validateInfo";
     const string validateResultName = "validateResult";

/**
```

9.18 Name binding

```
*/

     module NameBinding
     {
       /**
       This name binding is used to name the Software Distributor object to
       a ManagedElement object.
       */

       module SoftwareDistributor_ManagedElement
       {
          const string superiorClass = "itut_m3120::ManagedElement";
          const boolean superiorSubclassesAllowed = TRUE;
          const string subordinateClass =
           "itut_x744d1::SoftwareDistributor";
          const boolean subordinateSubclassesAllowed = TRUE;
          const boolean managerCreatesAllowed = TRUE;
          const DeletePolicyType deletePolicy =
              itut_x780::deleteContainedObjects;
          const string kind = "SoftwareDistributor";

       }; // module SoftwareDistributor_ManagedElement

       /**
       This name binding is used to name the Software Distributor object to
       a Subsystem object.
       */

       module SoftwareDistributor_Subsystem
       {
          const string superiorClass = "itut_x780::Subsystem";
          const boolean superiorSubclassesAllowed = TRUE;
          const string subordinateClass =
              "itut_x744d1::SoftwareDistributor";
          const boolean subordinateSubclassesAllowed = TRUE;
          const boolean managerCreatesAllowed = TRUE;
          const DeletePolicyType deletePolicy =
              itut_x780::deleteContainedObjects;
          const string kind = "SoftwareDistributor";

       }; // module SoftwareDistributor_Subsystem

       /**
       This name binding is used to name the Software Distributor object to
       a System object.
       */

       module SoftwareDistributor_System
       {
          const string superiorClass = "itut_x780::System";
          const boolean superiorSubclassesAllowed = TRUE;
          const string subordinateClass =
              "itut_x744d1::SoftwareDistributor";
          const boolean subordinateSubclassesAllowed = TRUE;
          const boolean managerCreatesAllowed = TRUE;
```

```
   const DeletePolicyType deletePolicy =
       itut_x780::deleteContainedObjects;
   const string kind = "SoftwareDistributor";

}; // module SoftwareDistributor_System

/**
This name binding is used to name the Software Unit object to
a Equipment object.
*/

module SoftwareUnit_Equipment
{
   const string superiorClass = "itut_m3120::Equipment";
   const boolean superiorSubclassesAllowed = TRUE;
   const string subordinateClass =
       "itut_x744d1::SoftwareUnit";
   const boolean subordinateSubclassesAllowed = TRUE;
   const boolean managerCreatesAllowed = TRUE;
   const DeletePolicyType deletePolicy =
       itut_x780::deleteContainedObjects;
   const string kind = "SoftwareUnit";

}; // module SoftwareUnit_Equipment

/**
This name binding is used to name the Software Unit object to
a ManagedElement object.
*/

module SoftwareUnit_ManagedElement
{
   const string superiorClass = "itut_m3120::ManagedElement";
   const boolean superiorSubclassesAllowed = TRUE;
   const string subordinateClass =
       "itut_x744d1::SoftwareUnit";
   const boolean subordinateSubclassesAllowed = TRUE;
   const boolean managerCreatesAllowed = TRUE;
   const DeletePolicyType deletePolicy =
       itut_x780::deleteContainedObjects;
   const string kind = "SoftwareUnit";

}; // module SoftwareUnit_ManagedElement

/**
This name binding is used to name the Software Unit object to
a Subsystem object.
*/

module SoftwareUnit_Subsystem
{
   const string superiorClass = "itut_x780::Subsystem";
   const boolean superiorSubclassesAllowed = TRUE;
   const string subordinateClass =
   "itut_x744d1::SoftwareUnit";
   const boolean subordinateSubclassesAllowed = TRUE;
   const boolean managerCreatesAllowed = TRUE;
   const DeletePolicyType deletePolicy =
   itut_x780::deleteContainedObjects;
   const string kind = "SoftwareUnit";

}; // module SoftwareUnit_Subsystem
```

```
/**
This name binding is used to name the Software Unit object to
a System object.
*/

module SoftwareUnit_System
{
   const string superiorClass = "itut_x780::System";
   const boolean superiorSubclassesAllowed = TRUE;
   const string subordinateClass =
       "itut_x744d1::SoftwareUnit";
   const boolean subordinateSubclassesAllowed = TRUE;
   const boolean managerCreatesAllowed = TRUE;
   const DeletePolicyType deletePolicy =
       itut_x780::deleteContainedObjects;
   const string kind = "SoftwareUnit";

}; // module SoftwareUnit_System

/**
This name binding is used to name the Executable Software object to
a Equipment object.
*/

module ExecutableSoftware_Equipment
{
   const string superiorClass = "itut_m3120::Equipment";
   const boolean superiorSubclassesAllowed = TRUE;
   const string subordinateClass =
       "itut_x744d1::ExecutableSoftware";
   const boolean subordinateSubclassesAllowed = TRUE;
   const boolean managerCreatesAllowed = TRUE;
   const DeletePolicyType deletePolicy =
       itut_x780::deleteContainedObjects;
   const string kind = "ExecutableSoftware";

}; // module ExecutableSoftware_Equipment

/**
This name binding is used to name the Executable Software object to
a ManagedElement object.
*/

module ExecutableSoftware_ManagedElement
{
   const string superiorClass = "itut_m3120::ManagedElement";
   const boolean superiorSubclassesAllowed = TRUE;
   const string subordinateClass =
       "itut_x744d1::ExecutableSoftware";
   const boolean subordinateSubclassesAllowed = TRUE;
   const boolean managerCreatesAllowed = TRUE;
   const DeletePolicyType deletePolicy =
       itut_x780::deleteContainedObjects;
   const string kind = "ExecutableSoftware";

}; // module ExecutableSoftware_ManagedElement
```

```
      /**
      This name binding is used to name the Executable Software object to
      a Subsystem object.
      */

      module ExecutableSoftware_Subsystem
      {
         const string superiorClass = "itut_x780::Subsystem";
         const boolean superiorSubclassesAllowed = TRUE;
         const string subordinateClass =
             "itut_x744d1::ExecutableSoftware";
         const boolean subordinateSubclassesAllowed = TRUE;
         const boolean managerCreatesAllowed = TRUE;
         const DeletePolicyType deletePolicy =
             itut_x780::deleteContainedObjects;
         const string kind = "ExecutableSoftware";

      }; // module ExecutableSoftware_Subsystem

      /**
      This name binding is used to name the Executable Software object to
      a System object.
      */

      module ExecutableSoftware_System
      {
         const string superiorClass = "itut_x780::System";
         const boolean superiorSubclassesAllowed = TRUE;
         const string subordinateClass =
             "itut_x744d1::ExecutableSoftware";
         const boolean subordinateSubclassesAllowed = TRUE;
         const boolean managerCreatesAllowed = TRUE;
         const DeletePolicyType deletePolicy =
             itut_x780::deleteContainedObjects;
         const string kind = "ExecutableSoftware";

      }; // module ExecutableSoftware_System

    }; // module NameBinding

/**
```

9.19 FileTypeConst module

```
*/

    module FileTypeConst
    {
      const string moduleName = "itut_x744d1::FileTypeConst";

      const short unstructuredText = 0; // FTAM-1
      const short unstructuredBinary = 1; // FTAM-3
      const short blockSpecial = 2;

    }; // end of module FileTypeConst

/**
```

9.20 DeliverResultConst module

```
*/

     module DeliverResultConst
     {
       const string moduleName = "itut_x744d1::DeliverResultConst";

       const short pass = 0;
       const short communicationError = 1;
       const short equipmentError = 2;
       const short qosError = 3;
       const short accessDenied = 4;
       const short notFound = 5;
       const short insufficientSpace = 6;
       const short alreadyDelivered = 7;
       const short inProgress = 8;
       const short unknown = 9;

     }; // end of module DeliverResultConst

}; // module itut_x744d1

#endif //_itut_x744_1_idl_
```

Printed in Switzerland
Geneva, 2003