DD CEN/TS 16238:2011

BSI Standards Publication

Postal services — Open Interface between Machine Control and Reading Coding System — MC/RC-Interface

raising standards worldwide™

National foreword

This Draft for Development is the UK implementation of CEN/TS 16238:2011.

This publication is not to be regarded as a British Standard.

It is being issued in the Draft for Development series of publications and is of a provisional nature. It should be applied on this provisional basis, so that information and experience of its practical application can be obtained.

Comments arising from the use of this Draft for Development are requested so that UK experience can be reported to the international organization responsible for its conversion to an international standard. A review of this publication will be initiated not later than 3 years after its publication by the international organization so that a decision can be taken on its status. Notification of the start of the review period will be made in an announcement in the appropriate issue of *Update Standards.*

According to the replies received by the end of the review period, the responsible BSI Committee will decide whether to support the conversion into an international Standard, to extend the life of the Technical Specification or to withdraw it. Comments should be sent to the Secretary of the responsible BSI Technical Committee at British Standards House, 389 Chiswick High Road, London W4 4AL.

The UK participation in its preparation was entrusted to Technical Committee SVS/4, Postal services.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

ISBN 978 0 580 73765 7

ICS 03.240; 35.240.60

Compliance with a British Standard cannot confer immunity from legal obligations.

This Draft for Development was published under the authority of the Standards Policy and Strategy Committee on 30 September 2011.

Amendments issued since publication

Date	Text affected

TECHNICAL SPECIFICATION

SPÉCIFICATION TECHNIQUE

TECHNISCHE SPEZIFIKATION

CEN/TS 16238

September 2011

ICS 03.240; 35.240.60

English Version

Postal services - Open Interface between Machine Control and Reading Coding System - MC/RC-Interface

Services postaux - Interface ouverte entre le système de Contrôle de la Machine et le système de Reconnaissance et de Codage - Interface MC/RC

Postalische Dienstleistungen - Offene Schnittstelle zwischen Maschinensteuerung und Lese- und Codier-System - MC/RC-Schnittstelle

EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

Management Centre: Avenue Marnix 17, B-1000 Brussels

Contents

Page

Foreword

This document (CEN/TS 16238:2011) has been prepared by Technical Committee CEN/TC 331 "Postal Services", the secretariat of which is held by NEN.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CEN [and/or CENELEC] shall not be held responsible for identifying any or all such patent rights.

According to the CEN/CENELEC Internal Regulations, the national standards organizations of the following countries are bound to announce this Technical Specification: Austria, Belgium, Bulgaria, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and the United Kingdom.

Introduction

There is a growing demand for postal operators to combine parts of their sorting automation equipment from different suppliers in order to optimise performance. In the past, this has led to project-specific interfaces being negotiated between one postal operator and one or multiple suppliers. These project-specific interfaces were developed by the suppliers and maintained for an agreed period. This approach has several disadvantages:

— The interface is derived from an interface that was not intended to be open.

— The interface is developed for a single project and works only in the context of that project (extra costs).

— Each participating supplier has to implement the interface (multiple effort).

— Experience shows that integration of components with project-specific interfaces is complex and expensive.

— Project-specific interfaces are not integrated into the product line and once the initially agreed maintenance period is over it may be difficult and expensive to maintain and/or may hinder the adoption of equipment upgrades.

This has led to “open interfaces” defined by one supplier. These still have the disadvantage of being in product use only by one supplier. Within a group of postal operators and suppliers it was decided to develop a set of “open standard interfaces” which will be developed by the suppliers and referred to by the postal operators. The benefits of these interfaces are expected to be that they:

— are fixed in an international standard (with change control);

— are agreed and implemented by major suppliers;

— are agreed by customers and therefore used in calls for tenders;

— will result in net savings with the high initial development effort and consequent higher basic equipment prices being more than offset by reduced project development, integration and maintenance costs;

— will minimize the need for project integration effort by reducing implementation timescales;

— will increase competition between suppliers by stimulating product improvements;

This Technical Specification covers the interface between an image controller (IC), the scanner subsystem including the scanner related image processing (IP) devices and the machine control (MC) subsystem of postal automation equipment. One related standard is CEN/TS 15448, *Postal Services – Open Standard Interface between image controller and enrichment devices (OCRs, video coding systems, voting systems)*.

Other work items (subject to agreement of CEN/TC 331 and the UPU Standards Board) will be defined to cover other areas as and when the need is identified and the resources for development become available. A separate project group for each interface will undertake the work.

1 Scope

This Technical Specification describes the “Open Standard Interface between Image Processor, Machine Control and Image Controller” (IP/MC/IC Interface) in the context of postal automation equipment.

The following architectural overview is the basis for this interface standardization:

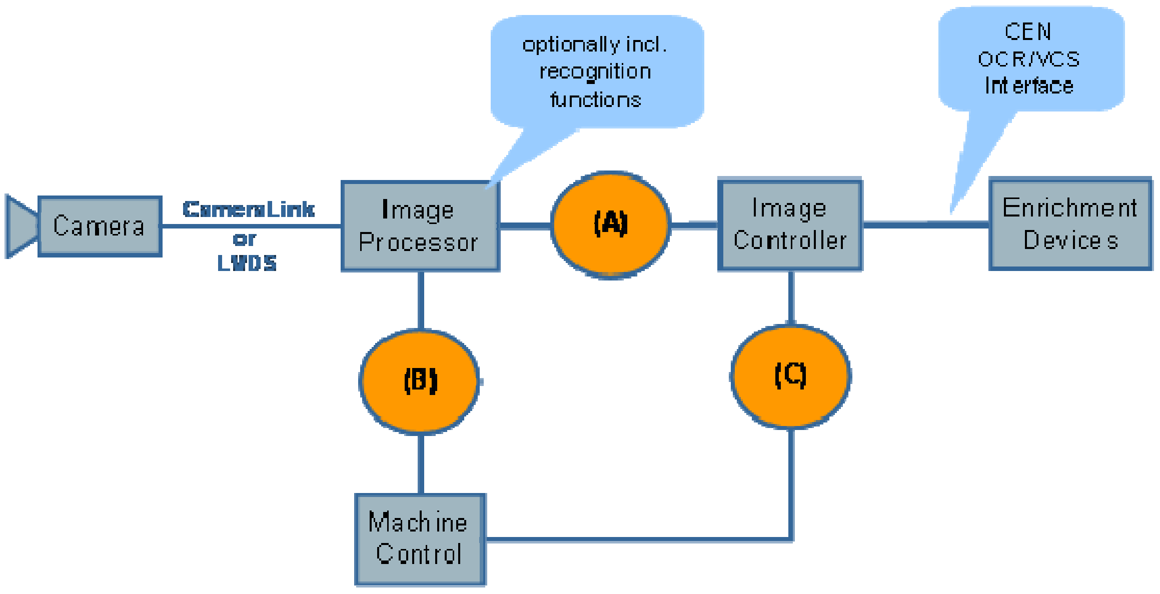


Figure 1 — System overview

It was agreed to unify the interfaces between

a) Image Processor and Image Controller,

b) Image Processor and Machine Control and

c) Machine Control and Image Controller

and to produce one common specification for this so-called **IP/MC/IC Interface**.

The communication partners of this interface will be called Machine or Machine Control (MC) on the one side and Reading/Coding (RC) System on the other side.

There may be several instances of this interface, depending on the implementation of the MC and the connected RC.

NOTE interfaces for synchronizing the lifted images with their mailpiece_IDs provided by the machine are not shown in the figure above and are not subject of standardization within the first release of this interface.

From the customer point of view, the following two scenarios are relevant. The systems MACHINE and RC SYSTEM are to be considered as “black boxes” thus not detailing internal system structure and interfaces.

1) The Machine already includes Camera and Image Processor and will be connected to a 3rd-party RC System including Image Controller and Enrichment Devices.

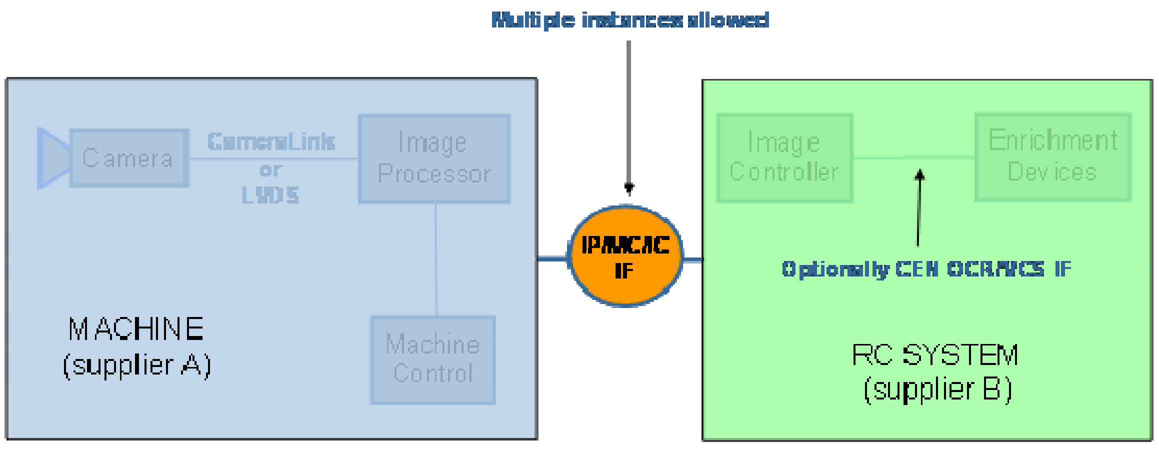


Figure 2 — System interfacing scenario 1

2) The Machine will be connected to a 3rd-party RC System including Camera, Image Processor, Image Controller and Enrichment Devices. The Camera and (possibly) the Image Processor will have to be mechanically integrated into the machine.

NOTE The camera can be provided by any 3rd-party. This should not impede on the IP/MC/IC interfaces !

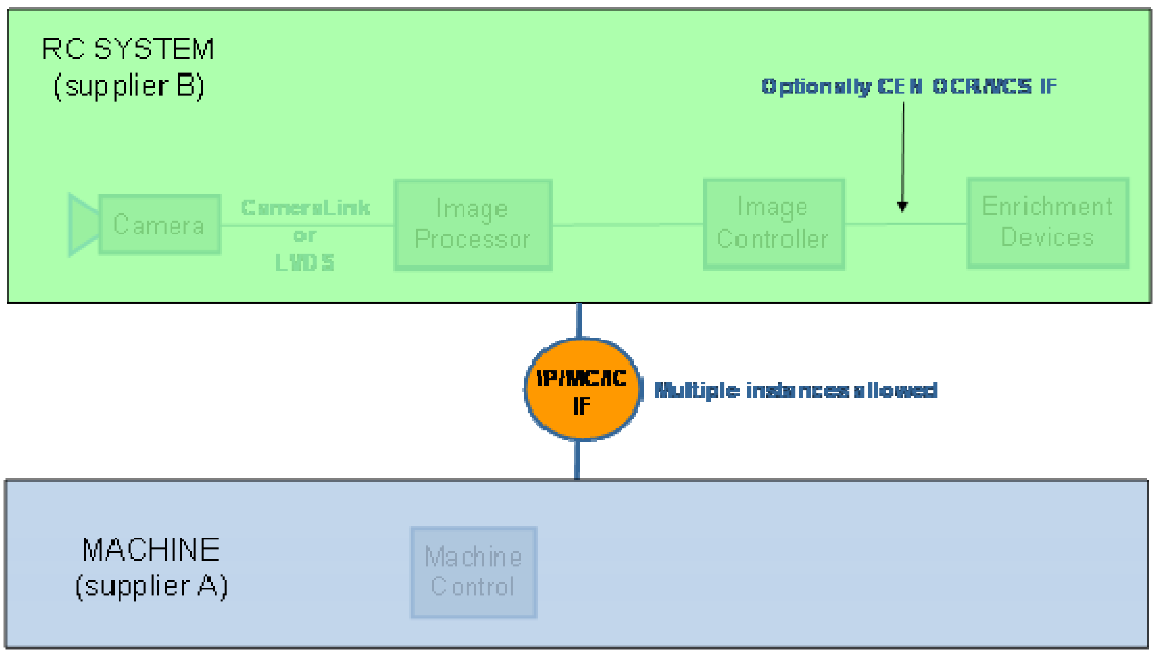


Figure 3 — System interfacing scenario 2

This standard is arranged under four main clauses as described in Figure 4.

— UCM (Use Case Model) describes the use cases for the IP/MC/IC Interface using sequence diagrams with messages.

— IDD (Interface Design Description) defines the data model for the IP/MC/IC interface.

— SDD (System Design Description) defines the mandatory specification of the IP/MC/IC interface in terms of architecture, services and behavioural models. In the Common Part of this clause no middleware or transport layer is specified. The common part of this clause is intended to be middleware-independent.

— SDD-TCP/IP, SDD-CORBA, SDD-SOAP in these specialized clauses. The specifications for three compatible transport solutions TCP/IP, CORBA and SOAP are provided. Further middleware solutions can be added when available, provided that they are fully compatible with the Common Part.

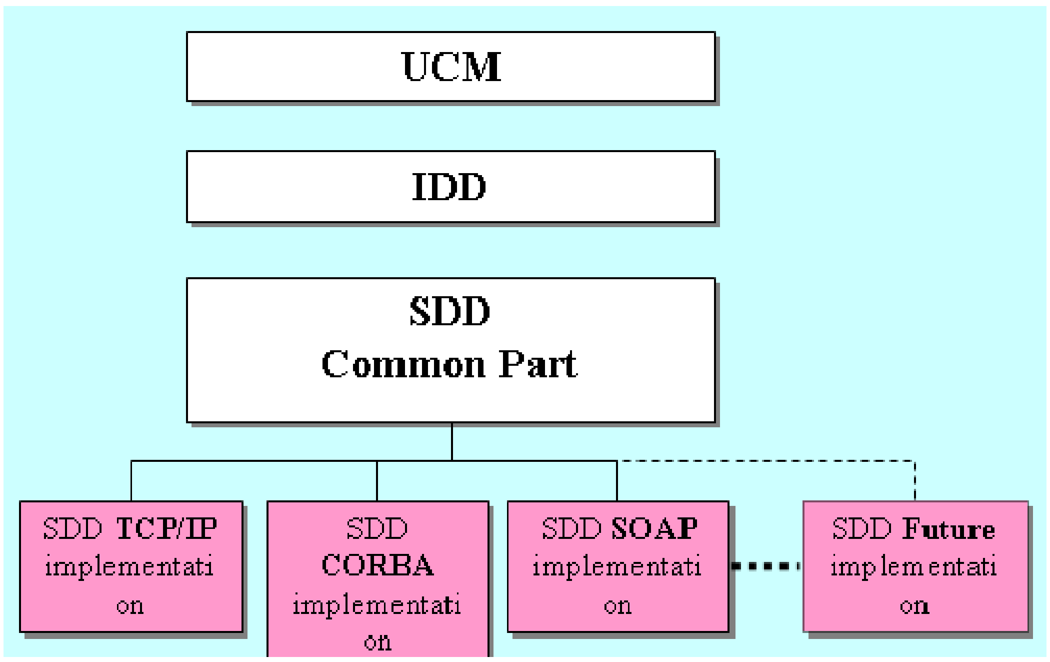

UCM
IDD
SDD
Common Part
SDD TCP/IP implementati on
SDD CORBA implementati on
SDD SOAP implementati on
SDD Future implementati on

Figure 4 — IP/MC/IC Interface Document Structure

2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, or references to a version number, only the edition cited applies. For undated references and where there is no reference to a version number, the latest edition of the referenced document (including any amendments) applies.

— CEN/TS 15448, *Postal Services – Open standard interface between image controller and enrichment devices (OCRs, video coding systems, voting systems)*

3 Terms and definitions

3.1
actor
coherent set of roles which users of uses cases play when interacting with these use cases. An actor has one role for each use case with which it communicates. See [1]

3.2
attributes
all non-image information related to a mailpiece

3.3
coding Desk
computer or terminal equipped with software to display images of mailpieces, and designed for a human operator (video coder) to enter information about the mailpiece

3.4
component
software Unit with a defined interface; might contain other components

3.5
data element
simple data type

3.6
data object
assembly of elements [1..*] and/or other data objects; recursive type

3.7
enrichment
process of generating new information about a mailpiece

NOTE Any information about the mailpiece may be used in this process, such as the image, image information or result data. The use of an image however, is not compulsory

3.8
enrichment device
system designed to enrich information about mailpieces

3.9
image
data acquired by the Image Processor (including scanner device) and stored as part of the mailpiece

3.10
image controller (IC)
the device that handles all the image data and results exchange between the transport and the enrichment devices – it is part of the RC system

3.11
image processor (IP)
the device that manages and processes images from the camera device – it may be in the Transport or in the RC system, depending on implementations

3.12
infrastructure data
the basic information, such as identification references which an Image Controller and Enrichment Device require in order to communicate effectively

EXAMPLE Letter ID, Submission ID.

3.13
machine control
the device - including software - that manages all transport functions inside postal automation equipment

3.14
mailpiece data
the information which describes attributes of the physical mailpiece which is used to aid and is a product of enrichment

EXAMPLE mailpiece width & height, indicia information, address location, city name, sort code etc.

3.15
offline
operational mode of a sorting machine, in which some processing of mail is done after the mail has been conveyed in the machine; there is a need for identification of individual mailpieces, to re-identify the mailpiece for which an additional sorting pass is required

NOTE Identification of mailpieces can be done by printing an ID-tag on individual mailpieces or by using print less identification means.

3.16
online
operational mode of a sorting machine, implying all the processing of mail is done while the mail is conveyed in the machine

NOTE Processing of mail may include sorting to output bins, printing of information on the mailpiece and other.

3.17
permanent error
fatal error as indicated by the middleware or application layer.

NOTE A non-fatal error may be considered to be a permanent error after repeated remedial handling

3.18
reading Coding System (RC System)
the system that provides image handling (the IC function) and retrieval of mailpiece data (ED function) from the images, depending on implementations

3.19
result
outcome of enrichment

3.20
street
street keying (street name and/or house number in street)

3.21
system
consists of components and the relationships between them (interfaces, communication)

4 Symbols and abbreviations

ED:	Enrichment Device (OCR, Video Coding, Voter system)
IC:	Image Controller
ID:	Identifier
IP:	Image Processor
IDD:	Interface Design Description

MC:	Machine Control
mpID:	Mailpiece Identification information
OCR:	Optical Character Recognition
PC:	Post Code
RC:	Reading (OCR) and Coding
ROI:	Region Of Interest
SDD:	System Design Description
UCM:	Use Case Model
UDDI:	Universal Description, Discovery and Integration
VCS:	Video Coding System
W3C:	World Wide Web Consortium
XML:	eXtendable Markup Language

5 Use Case Model (UCM)

This clause describes the use cases for the “Open Standard Interface between Image Processor, Machine Control and Image Controller” (IP/MC/IC Interface) using sequence diagrams with messages.

5.1 Conventions

Agreed syntax for use case names:

— Capital letter at the beginning of each word

— No underlines to connect the single words

EXAMPLE Submit Mailpiece

Agreed syntax for message names:

— No capital letter at the beginning of each word

— Underlines to connect the single words

EXAMPLE establish_connection

5.2 Use Case Overview

As already indicated in the system interfacing scenarios in Clause 1, there may be several instances of this interface between a Machine and the RC System. Each interface instance handles a point-to-point connection.

The use cases as listed in the table below and detailed in 5.3 describe the behaviour of an individual instance of this interface and do not need to consider any other instances.

Table 1

ID	Name	Description
UC01	Publish and Select Server	The RC System publishes its presence to a Server Selection service. The Machine uses the service to identify the RC System.
UC02	Connect	The Machine connects to the RC System. As part of connection protocol, the RC System indicates its capabilities. RC System capabilities (related to scenarios 1+2, see Clause 1) are, for example: ▪ Coding requests (with images) ▪ Result Storage ▪ Solicited result transmission to the Machine ▪ Unsolicited result transmission to the Machine ▪ Result requests based on ID-tag RC System capabilities (related to scenario 2, see Clause 1) are, for example: ▪ Coding requests (without images) ▪ Solicited result transmission to the Machine ▪ Barcode recognition requests ▪ Result requests based on ID-tag The capabilities of the RC System are checked against the expectations of the Machine. For example, in case the capabilities are not the expected ones, the Machine may disconnect and raise an alarm. The Machine indicates whether unsolicited results are expected or not for this connection.
UC03	Disconnect	The Machine or the RC System disconnects.
UC04	Submit Mailpiece	The Machine submits coding requests, with or without an image. The coding request identifies which attributes are expected and capabilities of the RC System to be used. There may be multiple submit_mailpiece messages for a single mailpiece (e.g. one request for id-tag reading, a later request for address reading).
UC05	Request Mailpiece Attributes	Request mail piece attributes from the tag database within the RC System.
UC06	Transmit Mailpiece Attributes	The RC System returns mailpiece attributes to the Machine, either in response to a Submit Mailpiece or a Request Mailpiece Attributes. The Transmit Mailpiece Attributes may also be returned on a different instance of the interface than the Submit Mailpiece was issued (e.g. on-line VCS results depending on machine architecture). There may be multiple responses of Transmit Mailpiece Attributes for a single “Submit Mailpiece” message, e.g. partial OCR results followed with final VCS result.
UC07	Update Mailpiece Attributes	The Machine reports new or changed attributes in order to update the mailpiece attribute database on the RC System (e.g. indicating the sort

ID	Name	Description
		bin). This use case is also used to indicate the events “Item Out Of Delay Line” and “Cancel Submit Mailpiece” to the RC System. Item Out Of Delay Line: The Machine indicates it has no further use for online OCR/VCS results for the specified mailpiece (e.g., mailpiece has reached the end of delay line). The RC System should avoid further online OCR/VCS processing. Cancel Submit Mailpiece: The Machine indicates it has no further use for the requested coding results for the specified mailpiece. The RC System shall avoid further OCR/VCS processing.
UC08	Request Image	The Machine requests an image from the RC System (incl. the IP). The image is used for batch ROI coordinates or for sample images for display.
UC09	Transmit Image	The RC System (incl. the IP) sends an image to the Machine, in response to Request Image.
UC10	Get RC System Status	The Machine requests status of the RC System.
UC11	Put RC System Status	The RC System indicates the status of its different capabilities to the Machine. In case of status change, this can also be unsolicited.
UC12	Get Machine Status	The RC System may request the status of the Machine (for example periodically).
UC13	Put Machine Status	The Machine indicates its status to the RC System, in response to a Get Machine Status (includes for example machine running, operation mode…). In case of status change, this can also be unsolicited.

5.3 Detailed Use Case Descriptions

General remarks for all use cases:

— The data structures shall be as close as possible to the ones of CEN/TS 15448.

— “mpID” means “unique mailpiece identifier” which may be an “online” mailpiece ID or an “offline” ID-tag.

— Messages drawn with dotted lines do not belong to the described use case but are included for better understanding.

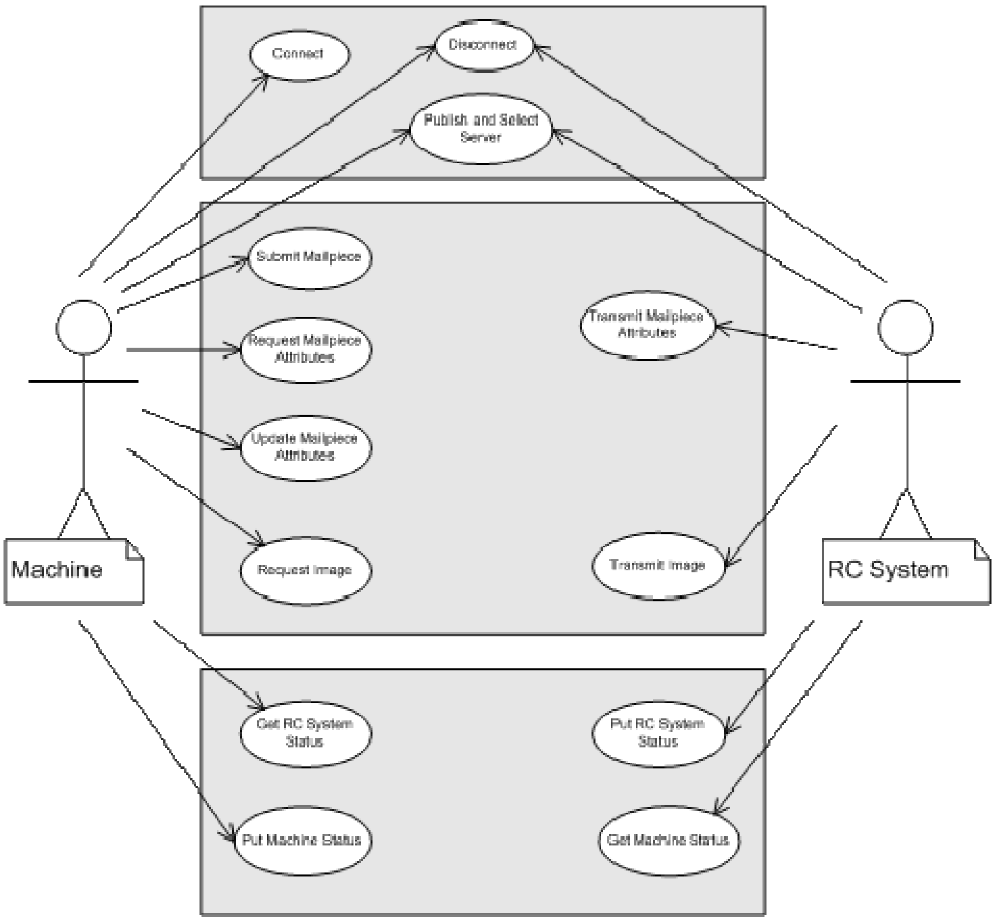


Figure 5 — Use case model

5.3.1 UC01 – Publish and Select Server

5.3.1.1 Brief Description

When the RC System is switched on and ready to be used by a Machine, it publishes its presence to a Server Selection service. Depending on the middleware chosen, the Server Selection service might be a UDDI service, Connection service, or simply a file. The Server Selection service will also provide a way to check for version compatibility. The Machine will use the Server Selection service to discover the RC System and retrieve the data necessary to establish a communication.

NOTE The RC System acts as a service and therefore publishes its presence to the Machine(s).

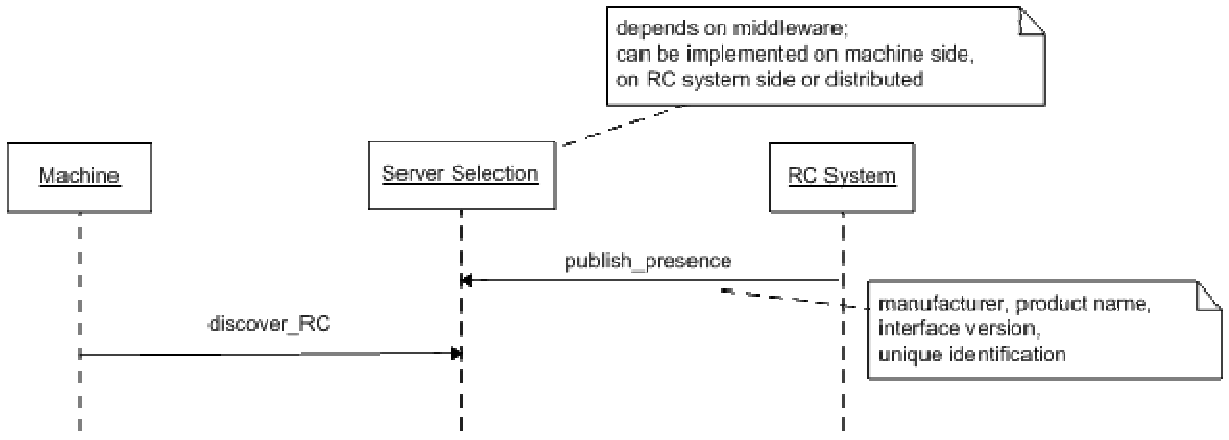


Figure 6 — Sequence diagram - Publish and Select Server

5.3.1.2 Actors

— RC System

— Machine

5.3.1.3 Flow of Events

5.3.1.3.1 Basic Flow

The basic flow covers the scenario where no exception handling is required. The Server Selection service for publication is available and can be reached by the RC System and the Machine.

1. The RC System starts and prepares itself to handle connection requests from Machines.

2. When the RC System is ready to handle connection requests, it publishes its presence to the Server Selection service.

3. The Machine uses the Server Selection service to discover the RC System. The Server Selection service returns a reference which can be used by the Machine to communicate directly with the RC System. The Machine may scan for available RC Systems or look explicitly for one.

The RC System is published when it is ready to accept a connection request.

5.3.1.3.2 AF01-01: RC System cannot register

If the RC System cannot register to the Server Selection service, it will attempt to register again after a configurable delay period.

5.3.1.3.3 AF01-02: Server Selection service already contains an entry for the RC System

This alternative flow is middleware dependent.

5.3.1.3.4 AF01-03: Server Selection service restarted

This alternative flow is middleware dependent.

When the Server Selection service is restarted, the references are not guaranteed to be available. There is no requirement that the Server Selection service is persistent. Therefore, the RC System must ensure that its current reference is entered correctly in the Server Selection service at all times.

5.3.1.4 Special Requirements

Table 2

SR ID	Description
SR01-01	If the RC System cannot register with the Server Selection service, it will re-attempt after a time defined by the supplementary requirement Publish_Retry_Delay. Registration attempts can be repeated indefinitely.

5.3.1.5 Pre-Conditions

Table 3

Pre ID	Description
PRE01-01	The RC System is configured with the information required to reach the Server Selection service (e.g. IP and Port number).
PRE01-02	The Server Selection service is running and is accessible by the RC System.

5.3.1.6 Post-Conditions

Table 4

Post ID	Description
POST01-01	The RC System is published in the Server Selection service.
POST01-02	The Machine knows the reference of the RC System. 11/4 should be used as the common reference size

5.3.2 UC02 – Connect

5.3.2.1 Brief Description

When a Machine wishes to use the services of a RC System which is published in the Server Selection service (see UC01 – Publish and Select Server), it connects to it as described by this use case. When this use case has been completed, the Machine can use the services provided by the RC System.

This use case enables a Machine and the RC System to communicate with each other.

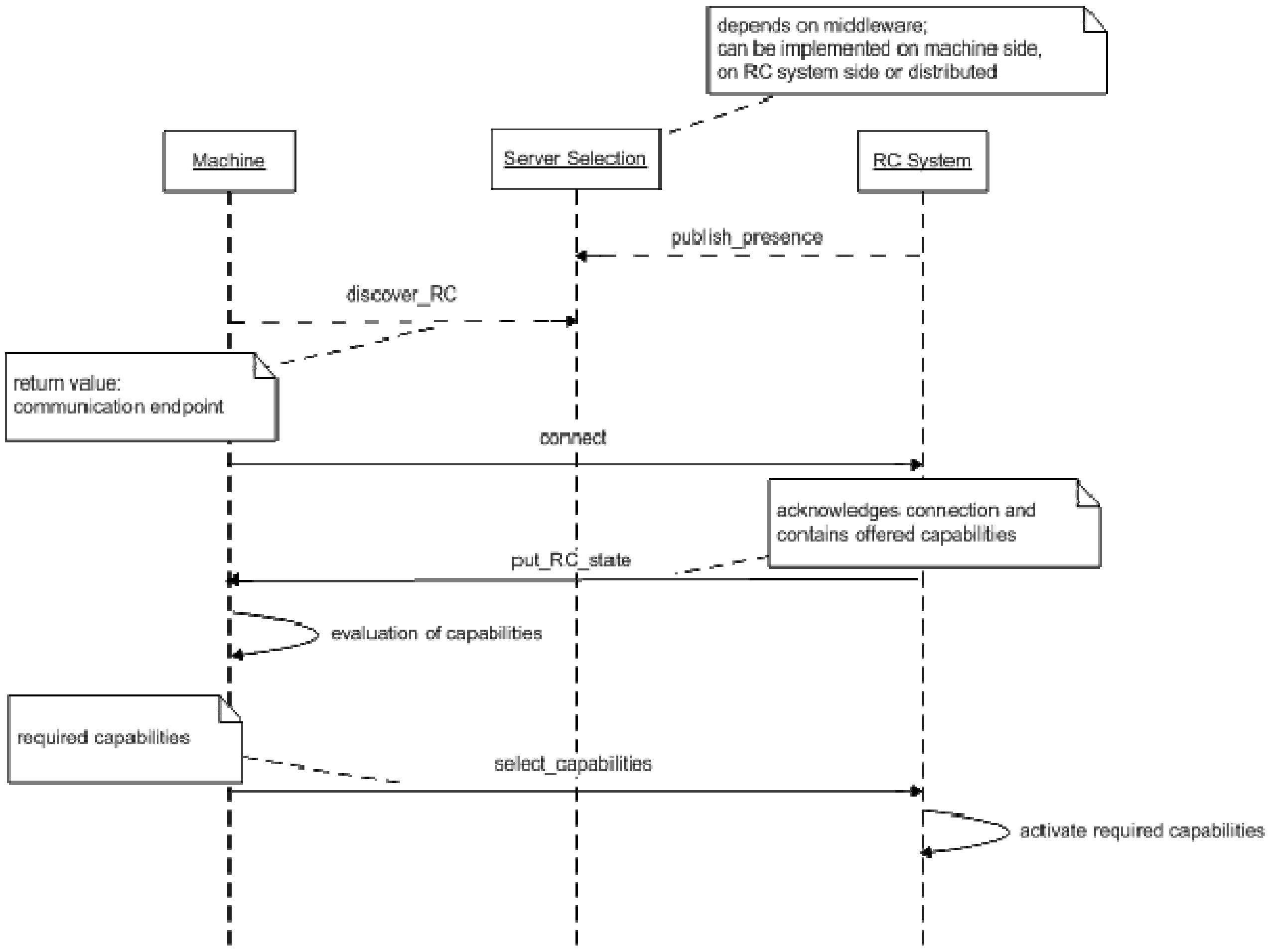

depends on middleware;
can be implemented on machine side,
on RC system side or distributed
Machine
Server Selection
RC System
publish_presence
discover_RC
return value:
communication endpoint
connect
acknowledges connection and
contains offered capabilities
put_RC_state
evaluation of capabilities
required capabilities
select_capabilities
activate required capabilities

Figure 7 — Sequence diagram – Connect

5.3.2.2 Actors

— Machine

— RC System

5.3.2.3 Flow of Events

5.3.2.3.1 Basic Flow

The basic flow describes the scenario when the Machine requests a connection and this is allowed by the RC System. As a precondition, the Machine has already selected an appropriate RC System.

1. The Machine requests a connection to the RC System.

2. The RC System checks the legality of the connection and acknowledges the connection by returning its state to the machine. This state information contains the capabilities of the RC System. These capabilities indicate what mailpieces can be processed by the RC System.

3. The Machine may restrict the set of capabilities to be used.

5.3.2.3.2 AF02-01: RC System rejects connection request

The RC System checks the legality of the connection and may reject the connection of the Machine for several reasons (e.g. licenses, too many connections, unsupported batch parameters …).

The rejection defines the reason for reject. The Machine may not attempt to reconnect before the time period defined by the supplementary requirement Connect_Retry_Delay.

Comment: Recovery is system dependent and not covered by this interface specification. However, please refer to the recommendations in 5.4.

5.3.2.3.3 AF02-02: Same Machine connects again to RC System

This is not considered as an alternative flow. Several connections are allowed between the same Machine and the RC System. The case of a Machine restart resulting in a reconnect is considered implementation specific.

5.3.2.3.4 AF02-03: Another Machine connects to RC System

Multiple Machines may connect to an RC System (each machine using its own instance of the interface). How the RC System handles multiple Machines is implementation specific and does not affect the interface.

5.3.2.3.5 AF02-04: Expected capability missing

If the Machine is waiting for a particular capability and the RC System cannot provide it, the Machine may or may not continue with the connection process.

Comment: Recovery is system dependent and not covered by this interface specification. However, please refer to the recommendations in 5.4. For example, the raise of an alarm depends on the Machine specification.

5.3.2.4 Special Requirements

Table 5

SR ID	Description
SR02-01	If the Machine receives at anytime a permanent error, it will determine again the reference of the RC System and the connection scenario will be re-performed.
SR02-02	A Machine may connect several times to the RC System using different instances of the interface.
SR02-03	More than one Machine may connect to the RC System using different instances of the interface. 11/4 should be used as the common reference size

5.3.2.5 Pre-Conditions

Table 6

Pre ID	Description
PRE02-01	UC01 has been successfully performed.
PRE02-02	All hardware is correctly installed (i.e. Each Machine and RC System has a unique IP address). Each Machine and RC System can reach the other via a network connection.

5.3.2.6 Post-Conditions

Table 7

Post ID	Description
POST02-01	The RC System knows the reference of the Machine.
POST02-02	The Machine knows of all the capabilities of the RC System.
POST02-03	The Machine is connected to the RC System
POST02-04	The Machine has committed to use only selected capabilities of the RC System.
POST02-05	The RC System grants to the Machine the selected capabilities.

5.3.3 UC03 – Disconnect

5.3.3.1 Brief Description

A disconnect may be initiated by the Machine or by the RC System, on an already existing connection.

Reasons for initiating disconnect include:

— explicit request from the Machine or from the RC System;

— software and hardware fault;

— transport-layer error;

— application-level error (like data out of bounds).

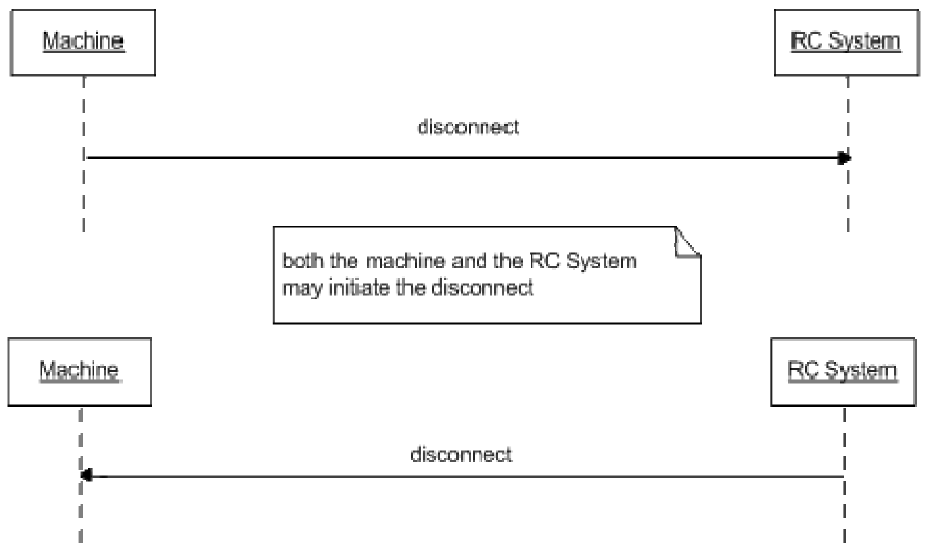

Machine
RC System
disconnect
both the machine and the RC System may initiate the disconnect
Machine
RC System
disconnect

Figure 8 — Sequence diagram – Disconnect

5.3.3.2 Actors

— Machine

— RC System

5.3.3.3 Flow of Events

5.3.3.3.1 Basic Flow

1. The Machine or the RC System requests a disconnect from the other system.

2. The Machine and the RC System are now disconnected.

5.3.3.3.2 AF03-01: Non-recoverable error occurs during Machine disconnect

If a non-recoverable error occurs whilst the Machine is executing the basic flow, the Machine will delete any objects which are required for communicating with the RC System. The RC System will receive non-recoverable errors when it attempts to communicate these objects and will also perform the necessary exception handling.

Comment: Recovery is system dependent and not covered by this interface specification. However, please refer to the recommendations in 5.4.

5.3.3.3.3 AF03-02: Non-recoverable error occurs during RC System disconnect

This alternative flow describes how the RC System disconnects from a machine. It may be executed for one, many or all connected machines and for all interface instances separately.

If a non-recoverable error occurs whilst the RC System is executing the above alternative flow, it will delete any objects which are required for the connection. The Machine will receive non-recoverable errors when it attempts to communicate these objects and will also perform the necessary exception handling.

Comment: Recovery is system dependent and not covered by this interface specification. However, please refer to the recommendations in 5.4.

5.3.3.4 Special Requirements

Table 8

SR ID	Description
SR03-01	No special requirement

5.3.3.5 Pre-Conditions

Table 9

Pre ID	Description
PRE03-01	Use case UC02 – Connect has been performed.

5.3.3.6 Post-Conditions

Table 10

Post ID	Description
POST03-01	The Machine and the RC System are no longer connected. All further communication requires that UC02 –Connect be performed.
POST03-02	The selected capabilities are released.

5.3.4 UC04 – Submit Mailpiece

5.3.4.1 Brief Description

The Machine Control executes this use case when it requires a mailpiece to be read by the RC System.

For the use case, Submit Mailpiece, there are two different sequence diagrams required depending on the system configuration: either the Image Processor belongs to the Machine or the Image Processor belongs to the RC System.

There may be one or more submit_mailpiece for a single mailpiece (also see Figure 9). This depends on the layout of the machine and the type of process devices in the machine.

5.3.4.1.1 Scenario 1: Image Processor belongs to Machine

The sequence of submit_mailpiece messages shown in the following figure is an example.

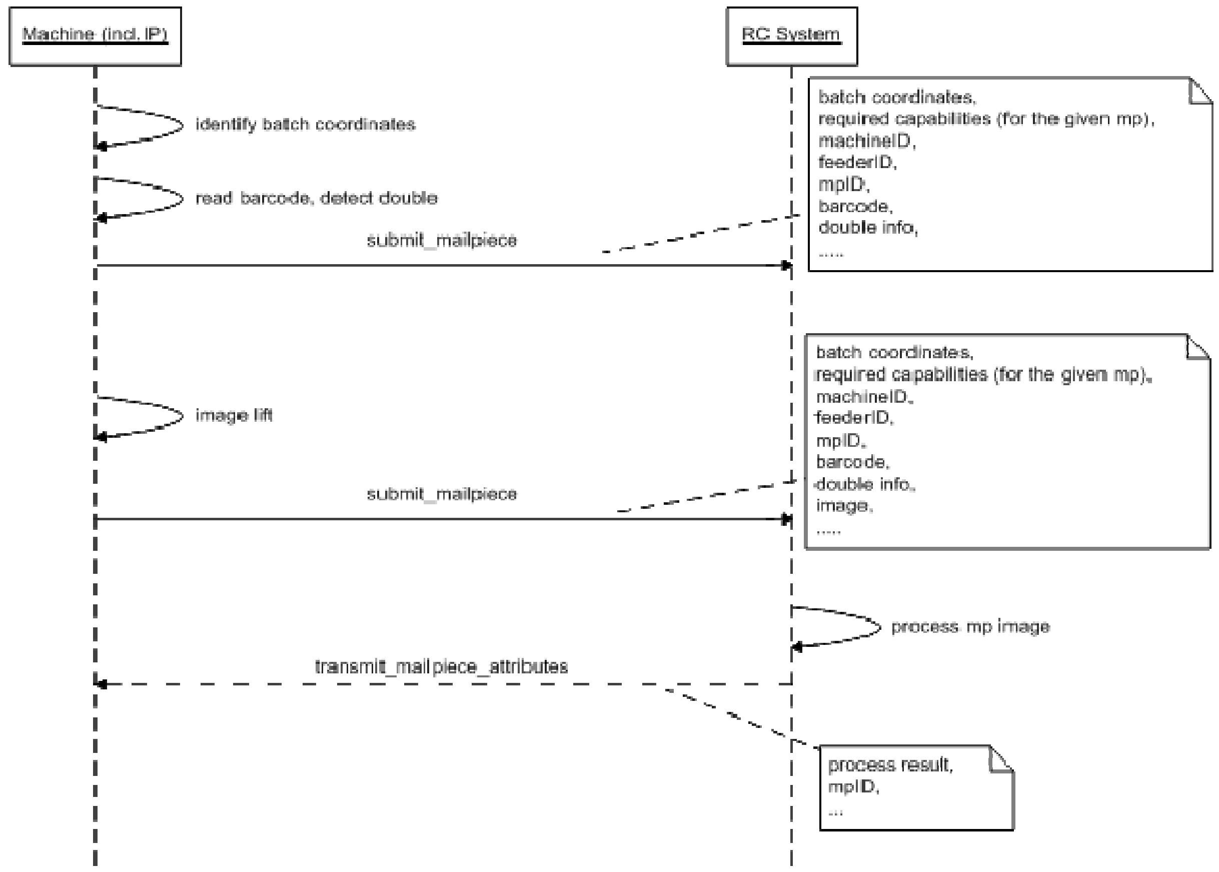

Machine (incl. IP)
RC System
identify batch coordinates
read barcode, detect double
submit_mailpiece
batch coordinates, required capabilities (for the given mp), machineID, feederID, mpID, barcode, double info, ...
image lift
submit_mailpiece
batch coordinates, required capabilities (for the given mp), machineID, feederID, mpID, barcode, double info, image, ...
process mp image
transmit_mailpiece_attributes
process result, mpID, ...

Figure 9 — Sequence diagram - Submit Mailpiece (scenario 1)

5.3.4.1.2 Scenario 2: Image Processor belongs to RC System

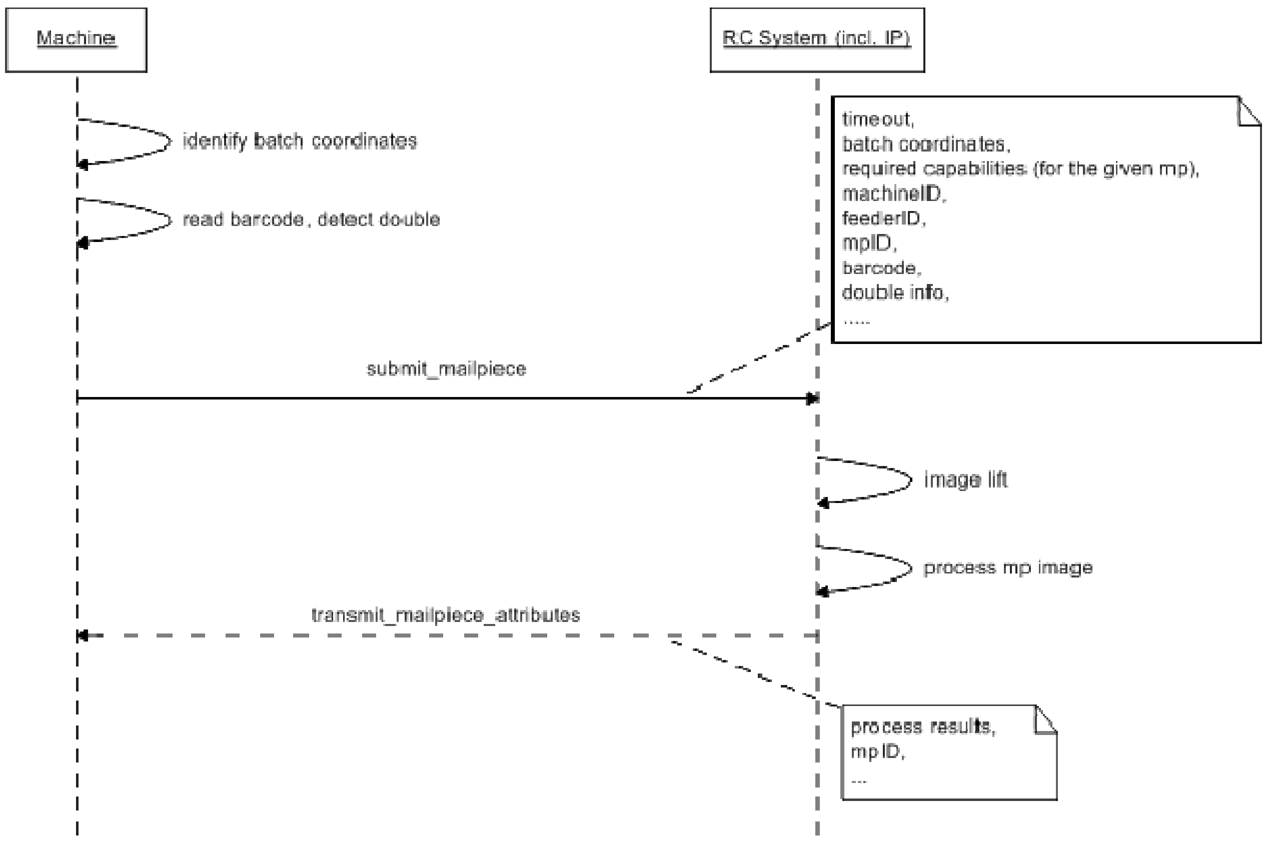


Figure 10 — Sequence diagram - Submit Mailpiece (scenario 2)

5.3.4.2 Actors

— Machine

— RC System

5.3.4.3 Flow of Events

5.3.4.3.1 Basic Flow

The basic flow describes the flow of events when the mailpiece can be submitted without error.

1. The Machine submits a mailpiece to the RC System. The submission must contain the following information:
   - Mailpiece Identification information (mpID) which uniquely identifies the submitted mailpiece for a sufficient period of time.
   - Requested Capabilities for the processing of the mailpiece.

- Timeout per requested capability within which the RC System has to return a result. The time in milliseconds within which the RC System has to generate a result. The time is relative to the time when the RC System receives the submission.

  At least the first execution of the use case for a certain mailpiece must furthermore contain the following information:

- Pre-knowledge (e.g. barcodes) and Infrastructure Data (e.g. machine Id)

2. The RC System derives the attributes for this mailpiece and returns them according to 5.3.6. For example, if the requested capabilities do not match those which were defined for the connection, or the Enrichment Device has previously indicated that it is no longer ready to receive any more work, the RC System will respond this error within 5.3.6.

5.3.4.3.2 AF04-01: Submission causes non-recoverable error

If the submission of a mailpiece causes a non-recoverable error, the Machine will attempt to close the connection (5.3.3) and to open a new one (5.3.2). Comment: Recovery is system dependent and not covered by this interface specification. However, please refer to the recommendations in 5.4.

5.3.4.4 Special Requirements

Table 11

SR ID	Description
SR04-01	The timeout for mailpiece processing starts when the submission is received by the RC System.
SR04-02	The mailpiece id alone identifies the mailpiece. It has to be unique at this interface for a sufficient period of time.

5.3.4.5 Pre-Conditions

Table 12

Pre ID	Description
PRE04-01	UC02 – Connect has been successfully executed.

5.3.4.6 Post-Conditions

Table 13

Post ID	Description
POST04-01	A mailpiece has been submitted to the RC System.

5.3.5 UC05 – Request Mailpiece Attributes

5.3.5.1 Brief Description

The Machine executes this use case to obtain result attributes for a mailpiece with a given mpID. In general, the Machine will execute this use case at most once per mailpiece.

When the use case is called for several mailpieces, the order of the resulting executions of 5.3.6 need not be in correspondence.

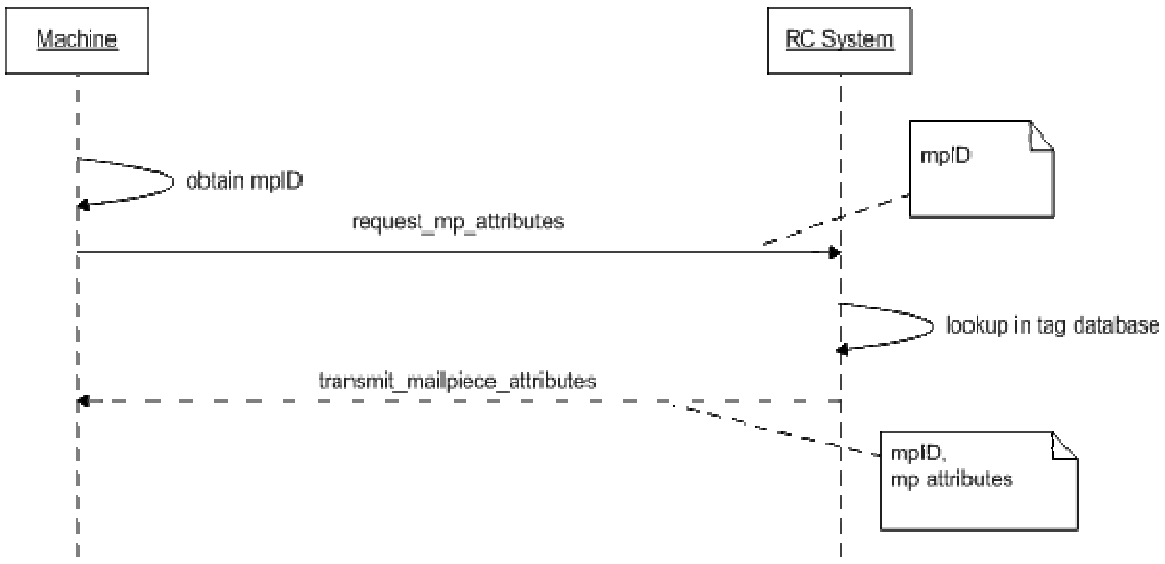


Figure 11 — Sequence diagram - Request Mailpiece Attributes

5.3.5.2 Actors

— Machine

— RC System

5.3.5.3 Flow of Events

5.3.5.3.1 Basic Flow

The basic flow describes the flow of events when the mailpiece can be submitted without error.

1. The Machine requests the result attributes for a mailpiece with given mpID. The request must contain the following information:
   - — Mailpiece Identification information (mpID) which uniquely identifies the submitted mailpiece for a sufficient period of time
   - — Timeout within which the RC System has to return a result. The time in milliseconds within which the RC System has to generate a result. The time is relative to the time when the RC System receives the request.
2. The RC System retrieves the result attributes for this mpID and returns them according to 5.3.6. If the given mpID (mailpiece) is not known in the underlying database, the RC System indicates this in 5.3.6.

If the RC System does not return a result in time, the Machine may take remedial action. This action is implementation specific and is not defined in further detail in this document.

5.3.5.3.2 AF05-01: Request causes non-recoverable error

If the request causes a non-recoverable error, the Machine should attempt to close the connection (5.3.3) and to open a new one (5.3.2). Comment: Recovery is system dependent and not covered by this interface specification. However, please refer to the recommendations in 5.4.

5.3.5.4 Special Requirements

Table 14

SR ID	Description
SR05-01	Results need not be returned in the order in which the mpIDs were submitted.
SR05-02	The mpID alone identifies the mailpiece. It has to be unique at this interface for a sufficient period of time.

5.3.5.5 Pre-Conditions

Table 15

Pre ID	Description
PRE05-01	UC02 – Connect has been successfully executed.

5.3.5.6 Post-Conditions

Table 16

Post ID	Description
POST05-01	A request for result attributes belonging to a given mpID has been transmitted to the RC System.

5.3.6 UC06 – Transmit Mailpiece Attributes

5.3.6.1 Brief Description

The RC System executes this use case to transmit result attributes to the Machine or to inform the Machine about reasons why the request cannot be processed or the result attributes cannot be transmitted.

A Transmit Mailpiece Attributes may occur as a response to 5.3.4 or to 5.3.5 or it may be unsolicited, e.g. for integrated barcode reading or for multiple Image Processors sending images on separate connections and a single machine control receiving all results on another connection.

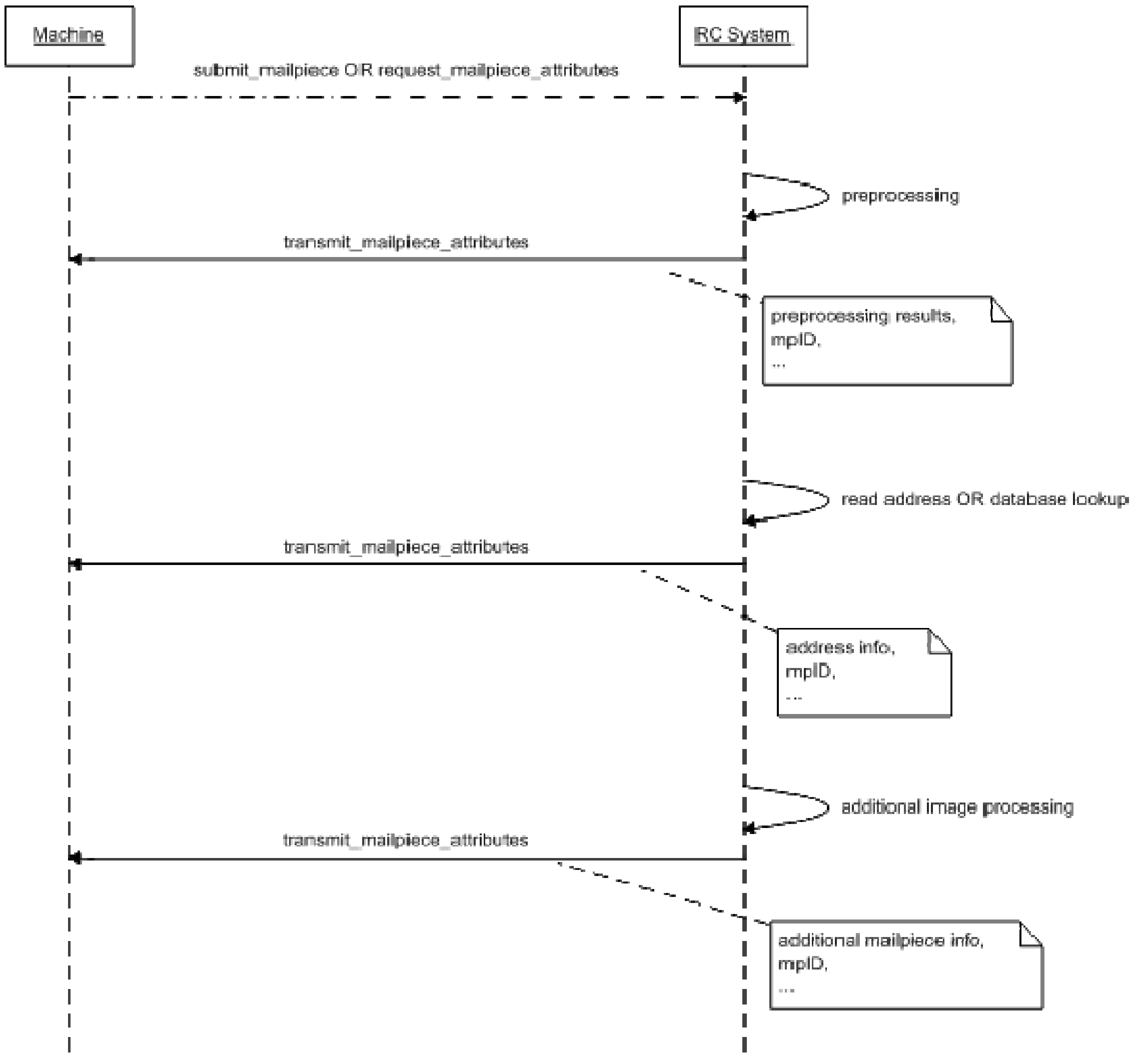

Machine
RC System
submit_mailpiece OR request_mailpiece_attributes
preprocessing
transmit_mailpiece_attributes
preprocessing results,
mpID,
...
read address OR database lookup
transmit_mailpiece_attributes
address info,
mpID,
...
additional image processing
transmit_mailpiece_attributes
additional mailpiece info,
mpID,
...

Figure 12 — Sequence diagram - Transmit Mailpiece Attributes

There may be one or more execution of this use case per mailpiece. For a number of mailpieces, for which 5.3.4 was called, the corresponding results may be transmitted out of sequence.

5.3.6.2 Actors

— RC System

— Machine

5.3.6.3 Flow of Events

5.3.6.3.1 Basic Flow

The basic flow describes the flow of events when the result attributes of a mailpiece are successfully obtained by the RC System and are transmitted in time.

1. The RC System transmits a result to the Machine. The result contains the following information:

   a. Mailpiece Identification
      information (mpID) which uniquely identifies the submitted mailpiece for a sufficient period of time.

   b. Result Information
      which can either be a complete result, or the best available result when a timeout occurs.

2. The RC System returns a result (mailpiece attributes) as soon as it is available. If a timeout is specified (by 5.3.4), any available (partial) result will be returned in time. For example, if an outward result is available but the inward result is not available, the outward information will be returned.

5.3.6.3.2 AF06-01: Transmission causes non-recoverable error

If the transmission causes a non-recoverable error, the RC System will attempt to close the connection (5.3.3) and assume that the Machine opens a new one (5.3.2).

Comment: Recovery is system dependent and not covered by this interface specification. However, please refer to the recommendations in 5.4.

5.3.6.3.3 AF06-02: Transmission unsuccessful

If the transmission of mailpiece attributes was unsuccessful, the behaviour of the RC System depends on its implementation.

Comment: Recovery is system dependent and not covered by this interface specification. However, please refer to the recommendations in 5.4.

5.3.6.3.4 AF06-03: Result not finalized within Timeout

If the RC System does not generate a final result for a mailpiece within a given Timeout, it will return a result before the expiration of the Timeout which indicates that the result generation has timed out. Any partial result information will be returned before the expiration of the Timeout. For example, if an outward result is available but the inward result is not available, the outward information will be returned.

5.3.6.4 AF06-04: No Result Returned

If the RC System does not return a result within a given Timeout, the Machine may take remedial action. This action is implementation specific and is not defined in further detail in this document.

5.3.6.5 Special Requirements

Table 17

SR ID	Description
SR06-01	The mailpiece id alone identifies the mailpiece. It has to be unique at this interface for a sufficient period of time.
SR06-02	A result is expected to be returned for a mailpiece within a given Timeout. If the mailpiece cannot be processed by the RC System, a result will be returned which indicates this.
SR06-03	Results need not be returned in the order in which the mailpieces were submitted
SR06-04	Results can be returned unsolicited. This is the case when several instances of the interface are used and there was a corresponding execution of UC05 – Request Mailpiece Attributes on another interface instance.

5.3.6.6 Pre-Conditions

Table 18

Pre ID	Description
PRE06-01	UC02 – Connect has been successfully executed

5.3.6.7 Post-Conditions

Table 19

Post ID	Description
POST06-01	A result has been transmitted to the Machine.

5.3.7 UC07 – Update Mailpiece Attributes

5.3.7.1 Brief Description

The Machine reports new or changed attributes in order to update the mailpiece attribute database on the RC System (e.g. indicating the sort bin). This use case is also used to indicate the following events to the RC System.

— Item Out Of Delay Line:

The Machine indicates it has no further use for online OCR/VCS results for the specified mailpiece (e.g., mailpiece has reached the end of delay line). For example, the RC System should avoid further online OCR/VCS processing.

— Cancel Submit Mailpiece:

The Machine indicates it has no further use for the requested coding results for the specified mailpiece. The RC System shall avoid further OCR/VCS processing.

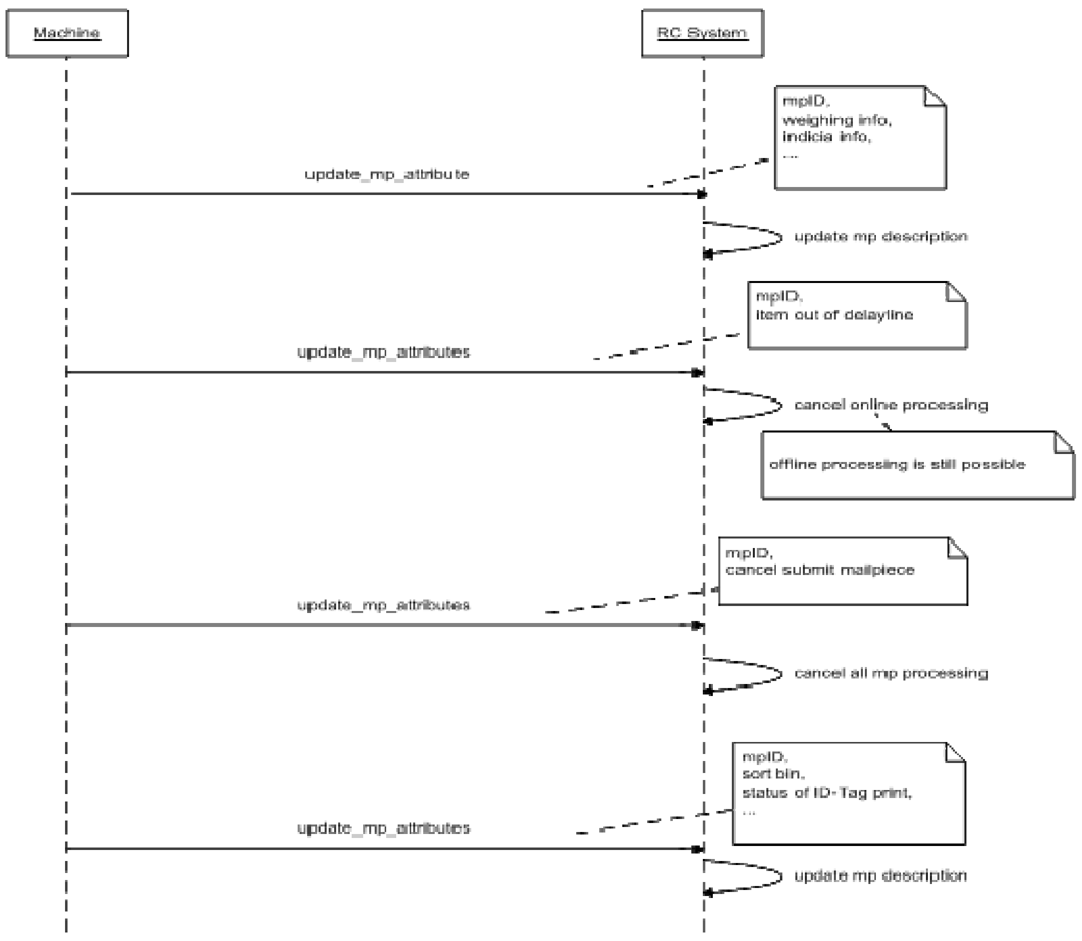


Figure 13 — Sequence diagram - Update Mailpiece Attributes

NOTE The sequence diagram above shows examples of alternative usages of the message update_mp_attribute. There is no fixed sequence of these messages.

5.3.7.2 Actors

— Machine

— RC System

5.3.7.3 Flow of Events

5.3.7.3.1 Basic Flow

The Machine sends result attributes for a mailpiece that shall be updated in the RC System. The request must contain the following information:

— Mailpiece Identification information (mpID) which uniquely identifies the submitted mailpiece for a sufficient period of time.

— Attributes of the mailpiece that shall be updated.

5.3.7.3.2 AF07-01: Update causes non-recoverable error

If the update of a mailpiece causes a non-recoverable error, the Machine should attempt to close the connection (5.3.3) and to open a new one (5.3.2).

Comment: Recovery is system dependent and not covered by this interface specification. However, please refer to the recommendations in 5.4.

5.3.7.3.3 AF07-02: Update with unknown mailpiece id

If the update contains a mailpiece id unknown to the RC System, then the RC System will ignore the update.

5.3.7.4 Special Requirements

Table 20

SR ID	Description
SR07-01	The mailpiece id alone identifies the mailpiece. It has to be unique at this interface for a sufficient period of time.

5.3.7.5 Pre-Conditions

Table 21

Pre ID	Description
PRE07-01	UC02 – Connect has been successfully executed.
PRE07-02	UC04 – Submit Mailpiece has been successfully executed for the same mpID before.

5.3.7.6 Post-Conditions

Table 22

Post ID	Description
POST07-01	Attributes for the mailpiece are updated in the RC System.

5.3.8 UC08 – Request Image

5.3.8.1 Brief Description

This use case supports applications for which a Machine must retrieve a mail piece from the RC System. More precise, the Machine can retrieve an image (set) of the most recently lifted image. Anticipated applications include:

— Displaying an image on Machine console to allow an operator to identify a Region-of-Interest for use in processing a large mailing of similar mail pieces.

— Continuous display of sampled images on Machine console to allow monitoring of image quality.

— Image and result logging performed by the Machine.

The Machine sends one request for each image. For each request, the RC System will execute use case 5.3.9, once. If several requests have been received from the Machine by the RC System, the order of images transmitted need not correspond to the order of requests.

This use case is not applicable if the Image Processor is part of the Machine.

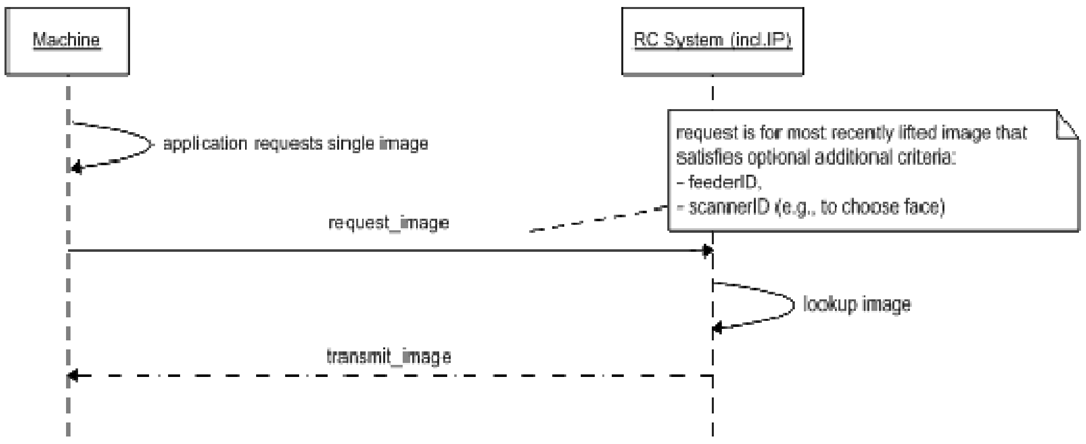

Machine
RC System (incl.IP)
application requests single image
request is for most recently lifted image that satisfies optional additional criteria:
- feederID,
- scannerID (e.g., to choose face)
request_image
lookup image
transmit_image

Figure 14 — Sequence diagram – Request Image

5.3.8.2 Actors

- Machine
- RC System

5.3.8.3 Flow of Events

5.3.8.3.1 Basic Flow

The steps for Request Image are:

1. The Machine requests an image from the RC System. The request may include the following optional parameters to indicate a specific image lift source:
   - feederID
   - scannerID (e.g., to choose face)
2. The RC System executes UC09 – Transmit Image.

5.3.8.4 Special Requirements

Table 23

SR ID	Description
---	---

5.3.8.5 Pre-Conditions

Table 24

Pre ID	Description
PRE08-01	UC02 – Connect has been executed successfully.

5.3.8.6 Post-Conditions

Table 25

Post ID	Description
POST08-01	A request for an image has been received by the RC System.

5.3.9 UC09 – Transmit Image

5.3.9.1 Brief Description

This use case is executed by the RC System in response to successful execution by the Machine of 5.3.8. The RC System looks up the most recently lifted image (set) satisfying the parameters of the request. The image and available attributes are transmitted to the Machine. If multiple images are transmitted, this can be done by using a set of TIFF containers (one per image) or a single multi-page TIFF container.

The transmitted data also includes parameters of the original request, so that the Machine can associate the image with the specified image lift device.

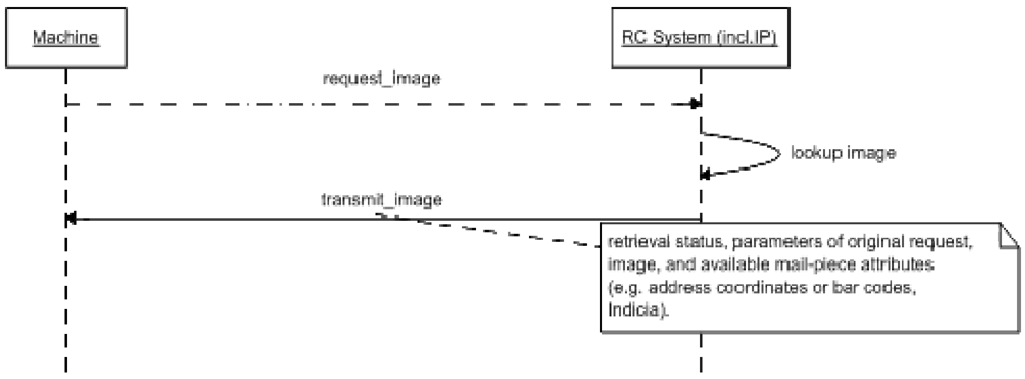

Machine
RC System (incl.IP)
request_image
lookup image
transmit_image
retrieval status, parameters of original request, image, and available mail-piece attributes (e.g. address coordinates or bar codes, Indicia).

Figure 15 — Sequence diagram – Transmit Image

NOTE if the machine detects, that the received image is corrupted, its behaviour is implementation dependent.

5.3.9.2 Actors

- RC System
- Machine

5.3.9.3 Flow of Events

5.3.9.3.1 Basic Flow

1. The RC System looks up the most recently lifted image satisfying parameters of the request (i.e., feederID, scannerID).

2. The RC System transmits the following to the Machine:

   — Retrieval status (indicating successful retrieval of the image).

   — Parameters from Request Image (i.e., feederID, scannerID).

   — Image.

   — Available attributes (e.g., address coordinates or bar codes, indicia).

5.3.9.3.2 AF09-01: Image not retrievable

If the RC System is unable to locate an image that satisfies parameters of the request, the RC System transmits the following to the Machine:

— Retrieval status (indicating failure to retrieve the image)

— Parameters from Request Image (i.e., feederID, scannerID).

5.3.9.4 Special Requirements

Table 26

SR ID	Description
---	---

5.3.9.5 Pre-Conditions

Table 27

Pre ID	Description
PRE09-01	Use case UC08 – Request Image has been successfully executed by the Machine

5.3.9.6 Post-Conditions

Table 28

Post ID	Description
POST09-01	Requested image and associated data has been transmitted from the RC System to the Machine.

5.3.10 UC10 – Get RC System Status

5.3.10.1 Brief Description

This use case is executed by the Machine at any time in order to retrieve the RC System status. This is to ensure that there is a connection between the Machine and the RC System.

This use case may be omitted if other communication events have been (successfully) carried out between the two devices during a configurable time period.

If the connection appears not to be working correctly, remedial action is taken by the Machine. This is described in the alternative flow of this use case at 5.3.10.3.2.

The RC System is aware of the execution of this use case. It uses this knowledge as a “ping” or “check alive” to ensure that it is still being used by the Machine. Should the RC System determine that it is not being used by the Machine, any required handling is implementation specific.

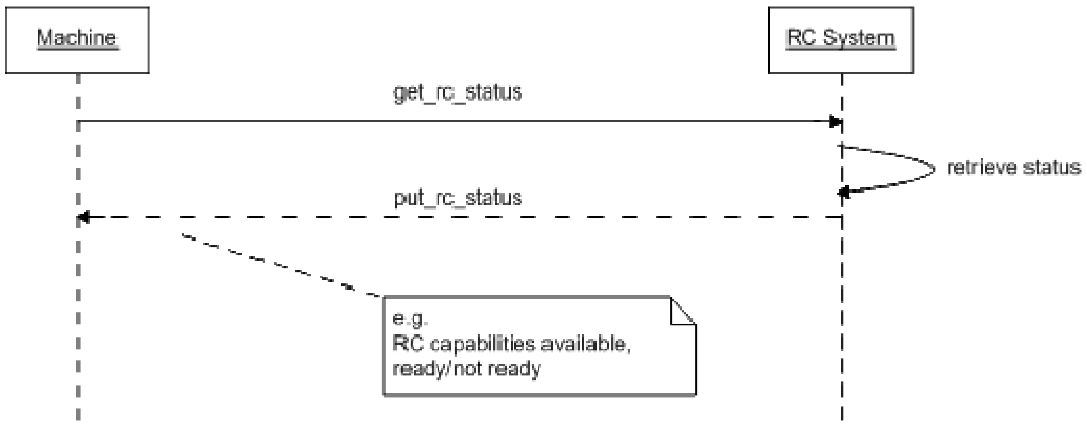


Figure 16 — Sequence diagram – Get RC System Status

5.3.10.2 Actors

— Machine

— RC System

5.3.10.3 Flow of Events

5.3.10.3.1 Basic Flow

1. The Machine requests the status of the RC System.
2. The RC System returns its status with the put_rc_status message.

NOTE If no status is returned within a configurable timeout, the Machine behaviour is implementation specific.

5.3.10.3.2 AF10-01: Non-recoverable error occurs

If a non-recoverable error occurs whilst the use case is being executed, the Machine behaviour is implementation specific.

Comment: Recovery is system dependent and not covered by this interface specification. However, please refer to the recommendations in 5.4.

5.3.10.4 Special Requirements

Table 29

SR ID	Description
SR10-01	There is a minimum period of time between two executions of this use case, which is defined in the configuration parameter *get_rc_status_period*

5.3.10.5 Pre-Conditions

Table 30

Pre ID	Description
PRE10-01	UC02 – Connect has been executed successfully.

5.3.10.6 Post-Conditions

Table 31

Post ID	Description
POST10-01	The Machine has received the status information from the RC System.

5.3.11 UC11 – Put RC System Status

5.3.11.1 Brief Description

This use case is executed by the RC System under the following condition(s):

— As an acknowledgement in the context of 5.3.2.

— In response to a get_rc_status issued by the Machine.

— On RC System status change: any status change in the RC System is promptly notified to all the connected Machines.

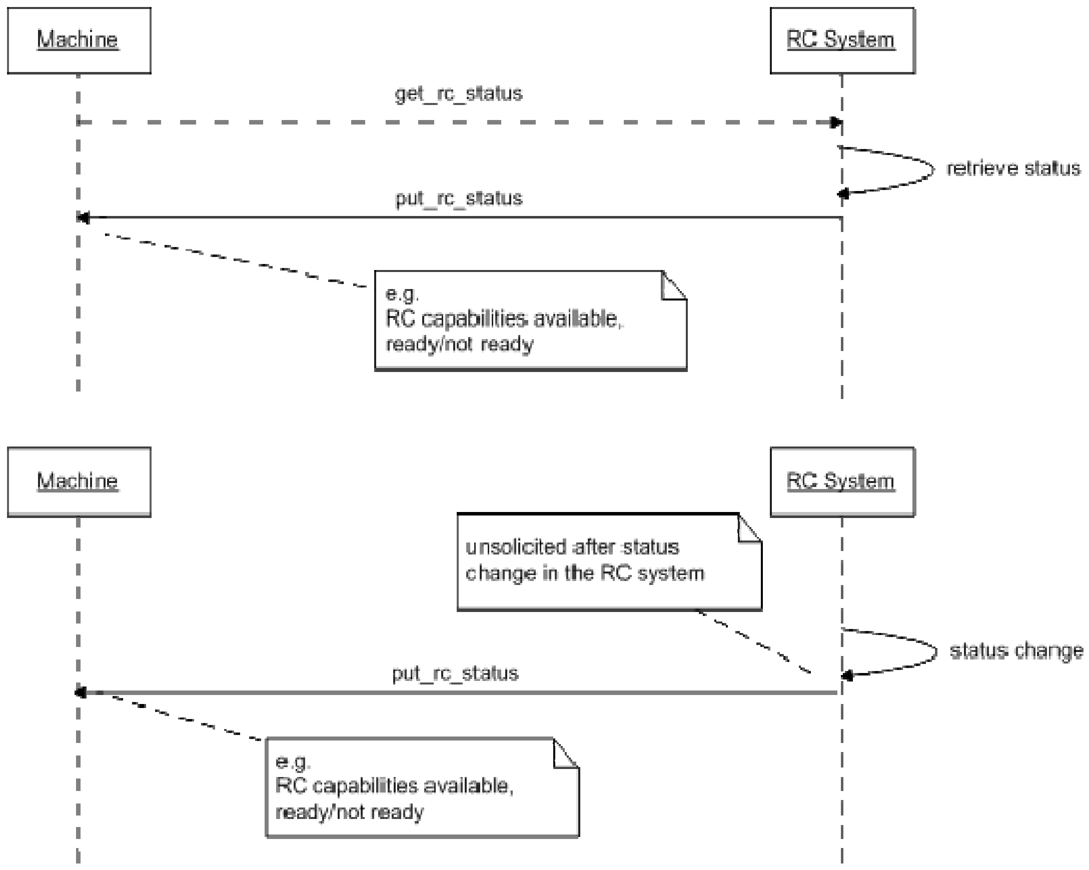


Figure 17 — Sequence diagram – Put RC System Status

5.3.11.2 Actors

— RC System

— Machine

5.3.11.3 Flow of Events

5.3.11.3.1 Basic flow

The RC System issues a put_rc_status in the event one of the following occurs:

1. A Machine connects to the RC System.
2. A status change of the RC System itself or of one of its declared capabilities (unsolicited).
3. The Machine requests the status from the RC System using the get_rc_status.

The status information returned by the RC System is fully described in the IDD part of this interface definition.

5.3.11.3.2 AF11-01: Non-recoverable Error Occurs

If a non-recoverable error occurs whilst the use case is being executed, the RC System behaviour is implementation specific.

Comment: Recovery is system dependent and not covered by this interface specification. However, please refer to the recommendations in 5.4.

5.3.11.4 Special Requirements

Table 32

SR ID	Description
SR11-01	This use case executes "immediately", and causes the RC System to send its status information.

5.3.11.5 Pre-Conditions

Table 33

Pre ID	Description
PRE11-01	Use Case UC02 – Connect has been executed
PRE11-02	Either the RC System received a get_rc_status or the status of the RC System changed

5.3.11.6 Post-Conditions

Table 34

Post ID	Description
POST11-01	The Machine has received the RC System status.

5.3.12 UC12 – Get Machine Status

5.3.12.1 Brief description

This use case is executed by the RC System at any time in order to retrieve the status of a connected Machine. This is to ensure that there is a connection between the RC System and the Machine.

This use case may be omitted if other communication events have been successfully carried out between the two devices during a configurable time period.

If the connection appears not to be working correctly, remedial action is taken by the RC System. This is described in the alternative flow of this use case in 5.3.12.3.2.

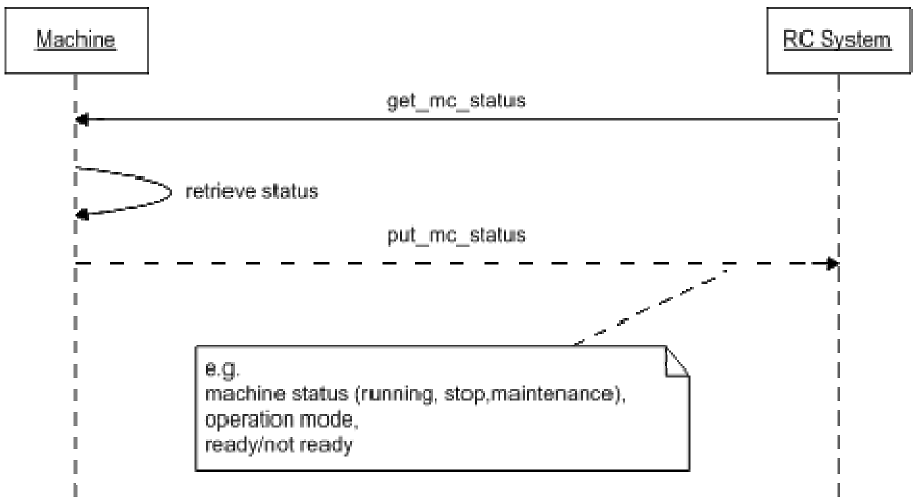


Figure 18 — Sequence diagram – Get Machine Status

5.3.12.2 Actors

— RC System

— Machine

5.3.12.3 Flow of Events

5.3.12.3.1 Basic Flow

1. The RC System requests the status of the Machine.
2. The Machine returns its status with the put_mc_status message.

NOTE If no status is returned within a configurable timeout, the RC System behaviour is implementation specific.

5.3.12.3.2 AF12-01: Non-recoverable error occurs

If a non-recoverable error occurs whilst the use case is being executed, the RC System behaviour is implementation specific.

Comment: Recovery is system dependent and not covered by this interface specification. However, please refer to the recommendations in 5.4.

5.3.12.4 Special Requirements

Table 35

SR ID	Description
SR12-01	There is a minimum period of time between two executions of this use case, which is defined in the configuration parameter *get_mc_status_period*

5.3.12.5 Pre-Conditions

Table 36

Pre ID	Description
PRE12-01	UC02 – Connect has been executed successfully

5.3.12.6 Post-Conditions

Table 37

Post ID	Description
POST12-01	The RC System has received the status of the Machine.

5.3.13 UC13 – Put Machine Status

5.3.13.1 Brief Description

This use case is executed by the Machine under the following condition(s):

— In response to a get_mc_status issued by the RC System.

— On Machine status change: any status change in the Machine is promptly notified to the connected RC System.

This use case is executed only after connection between the Machine and the RC System.

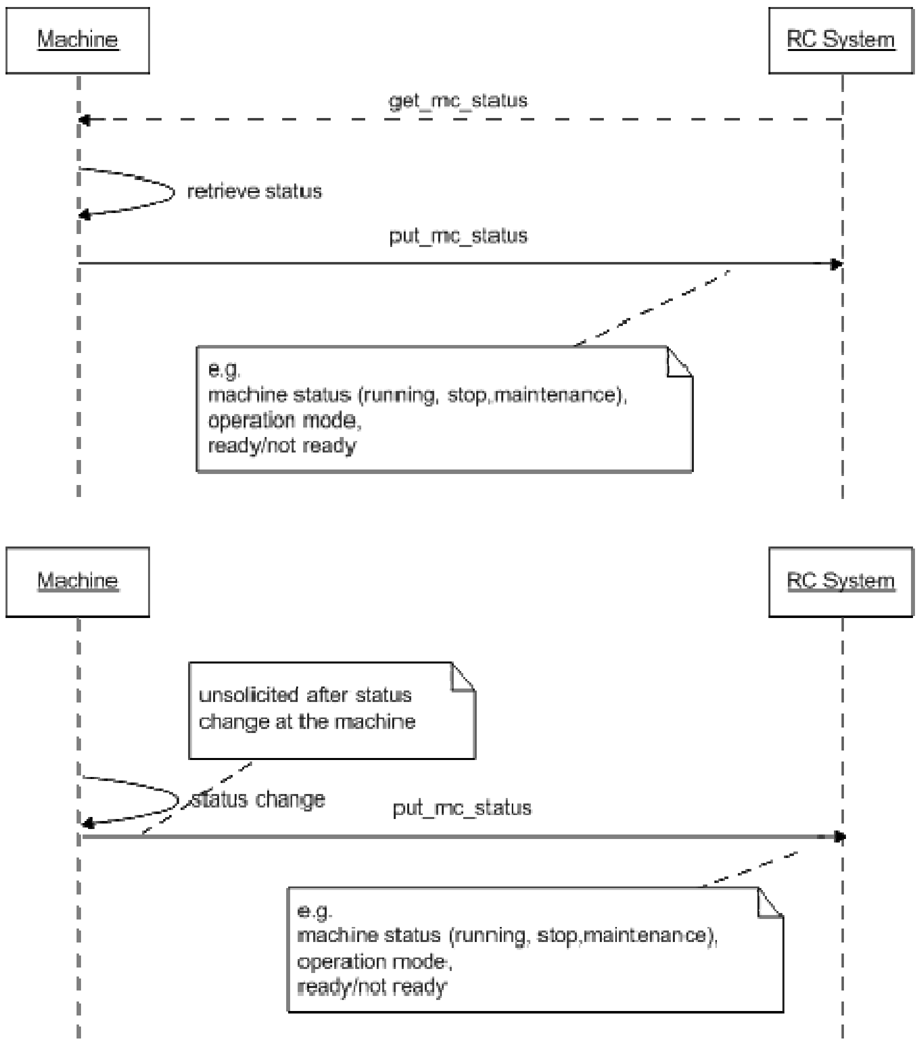


Figure 19 — Sequence diagram – Put Machine Status

5.3.13.2 Actors

— Machine

— RC System

5.3.13.3 Flow of events

5.3.13.3.1 Basic flow

The Machine issues a put_mc_status in the event one of the following occurs:

1. A status change of the Machine itself (unsolicited).

2. The RC System requests the status from the Machine using the get_mc_status.

The status information returned by the Machine is fully described in the IDD part of this interface definition.

5.3.13.3.2 AF13-01: Non-recoverable error occurs

If a non-recoverable error occurs whilst the use case is being executed, the Machine behaviour is implementation specific.

Comment: Recovery is system dependent and not covered by this interface specification. However, please refer to the recommendations in 5.4.

5.3.13.4 Special requirements

Table 38

SR ID	Description
SR13-01	This use case executes "immediately" and causes the Machine to send its status information.

5.3.13.5 Pre-conditions

Table 39

Pre ID	Description
PRE13-01	Use Case UC02 – Connect has been executed
PRE13-02	Either the Machine received a get_mc_status or the status of the Machine changed

5.3.13.6 Post-conditions

Table 40

Post ID	Description
POST13-01	The RC System has received the Machine status.

5.4 General error handling

In order to avoid problems at a later stage of mail processing (for example that mail cannot be sorted because no results are available) it is important that the appropriate actions are taken immediately when errors occur.

This concerns the following error groups:

— Connection problems,

— Unavailable functionality/capability of the Machine,

— Unavailable functionality/capability of the RC System.

In the previous use case description, where it was noted that error handling shall be implementation specific, the following recovery steps are recommended.

1. Retry the failed operation for a configurable number of times.
2. If the previous step was not successful, retry the Disconnect/Connect sequence for a configurable number of times.
3. If the previous step was not successful, ensure a non-recoverable error is displayed for operator.

Errors on system level shall be handled middleware specific. Errors on application level shall be reflected within the XML data that is exchanged between the MC and the RC System and defined in the IDD part of this interface.

6 Interface Design Description (IDD)

This clause provides XML schemas that define the data model used for the definition of an interface in the postal coding environment. The focus of this model is on the interface between a Reading and Coding System (RC) and a Machine Controller (MC). Either the RC or the MC contains a Scanner (refer to chapter 1).

An XML schema is a formalization of specific constraints, expressed as rules or a model of structure that applies to a (class of) XML documents. This applies as mentioned above to the static structure of a data type as well as to the content of single elements.

XML schemas provide a wide range of features to specify data structures. They can be used to validate XML instance documents. This is the most common use and generally the initial purpose of schemas.

This clause defines the syntax and partly the semantics of the XML data exchanged in the interface. Using the definition contained herein, it is possible to enable correct interaction between an MC and an RC. However, the data will not be fully checked for semantic correctness (e.g. is the postcode 4 or 5 digits). Some of the relevant semantic definitions of the interface are customer specific and outside the scope of the standard.

The intended audience is any manufacturer of an MC or RC who implements this standard interface.

The fundamental prerequisite is knowledge about XML and XML schemas. It is definitely necessary to understand the data model. Refer to A.1 to understand XML schemas.

Refer to Clause 5 to get information about the Use Cases and Requirements, which affects the Domain Data Model and its specifications.

6.1 TIFF Definition

6.1.1 Tiff Usage

TIFF file format and usage will be as defined in CEN/TS 15448.

When several images are sent with one message, these images can either be contained in one multi page TIFF file or in a set of TIFF files.

The middleware dependent specifications describe how the tiff images are transferred by the services Submit Mailpiece and Transmit Image.

6.2 Mailpiece Data Definition

6.2.1 Requirements

This clause describes the requirements for the data domain model.

6.2.1.1 Filtering of Data

MC and RC import data objects. For each component's processing needs there is relevant data to be extracted from the data object. The data object may contain elements which are irrelevant to the receiving party for the further processing. Two issues have to be distinguished:

1) If MC and RC work with different versions of the Data Model and one version is extended and hosts new elements. The component which received the extended object has to filter the data object. Unknown elements have to be ignored without exception.

2) Both components work with the same version of the Data Model, but the receiver gets a data object which hosts information not required for the processing. The component extracts the relevant data elements.

6.2.1.2 Non-Functional Requirements

6.2.1.2.1 Country-Specific Content

The data format must provide the possibility to exchange definitions of country specific content.

Address information can vary from country to country, so it must be possible to exchange only the address definition without having to adapt the definition of other related objects.

6.2.1.2.2 Platform independent Data Format

The data format must be defined platform independently.

6.2.1.2.3 Standard Format

It shall not use a proprietary definition of the data format but standards have to be applied. The processing of the data modelled by the data format must be supported by standard tools.

6.2.1.2.4 Performance

To enable high performance of the system using the data format, it is required to keep the amount of data that has to be stored and transmitted between the different system components as low as possible.

6.2.1.2.5 Data Types

The following data types must be supported by the data format:

1) **Simple Data Types** like e.g. strings, numbers, time, date. It must also be possible to define the allowed value ranges for simple data types.

2) **Complex Data Types**: Data types that group members of Simple Data Types or other Complex Data Types. These groups of data items are logical constructs that are a container for data items that belong together.

6.2.1.2.6 Versioning

6.2.1.2.6.1 Rules for Version Tags

a) There must be an unambiguous rule for how version tags are to be created and assigned

b) A version tag must have the following format:

Major-ID.Minor-ID with fixed length of the minor ID. The length is set to 2.

EXAMPLE 0.01, 1.13, 2.00.

6.2.1.2.6.2 Compatibility

Backward compatibility of the data format is to be achieved. It must be ensured that components adapted to a newer version of the data format can work on an older version of the data format as well. This requirement affects versions of the data format which exclusively differ in the minor version.

The data format and the guidelines for evolving new versions of the data format have to create the basic conditions to make it possible to fulfil the above requirements.

6.2.1.2.7 Namespaces

There must be rules defined how namespace names are built. This is required to avoid naming conflicts and to build packages.

6.2.1.2.8 General Unique ID

There is one unique ID used within the interface:

— a mailpiece ID, the final, unique ID to be associated with a mailpiece and its task and image data. This ID may or may not be the ID-Tag.

It is prohibited to change the ID during processing. The processing encloses the time of all transactions about the mail item, i.e. from the first submission to the transmission of the final result.

NOTE The requirement does not address the ID-Tag printed on the mailpiece. The ID-Tag is tracked as a separate attribute.

6.2.1.2.9 Data Content: Pre-Knowledge

Pre-Knowledge information must be transmitted via the data format from the MC to the RC.

Dependent on the system configuration Pre-Knowledge information is available that has to be assigned to the RC as input data. Pre-knowledge can be classified as follows:

— Hints from an operator like the position of the address label on the scanned image;

— Barcode reader results;

— Double-feed detection results;

— On-board address recognition results.

6.2.2 Model Commitments

6.2.2.1 Namespaces

Namespaces provide a mechanism that helps to differentiate XML documents. W3C XML Schema associates a namespace to all objects (elements, simple and complex types) defined in a schema. The namespaces are URI based.

Namespaces should be named meaningful and according to a scheme that avoids naming conflicts. In addition, the proposed convention for defining namespace identifiers makes it possible to get information about the organization or provider that created the elements of the namespace, the product or the project for which the namespace and its elements had been defined.

Namespace identifier has to consider:

- creation date of the schema;
- project or product name;
- provider domain.

Recommended convention for namespace identifiers:

http://<provider_domain>/Industry/<creation_date>/PostalAutomation/<schema_name>/

<provider_domain>	domain of the provider, e.g. www.cen.com.
<creation_date>	the date of the creation of the schema version: yyyy/mm e.g. “2003/09”.
<schema_name>	the name of the schema.

Example for a released version: http://www.cen.com/Industry/2008/06/PostalAutomation/Base

Example for a working draft: http://www.cen.com/Industry/2008/06/draft/PostalAutomation/Base

6.2.2.1.1 Using a Shortcut instead of the URI Reference

XML Schema provides the use of shortcuts, which can be used instead of the URI references.

EXAMPLE

```
<?xml version="1.0" encoding="UTF-8"?>
 <schema targetNamespace="http://www.cen.com/Industry/2008/06/PostalAutomation/base"
          xmlns:mcrc=" http://www.cen.com/Industry/2008/06/PostalAutomation/base "
          xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" version="0.1">
```

The W3C schema specific namespace is **xmlns=http://www.w3.org/2001/XMLSchema.**

The domain namespace is specified as follows:
targetNamespace=http://www.cen.com/Industry/2008/06/PostalAutomation/base.

xmlns:mcrc="http://www.cen.com/Industry/2008/06/PostalAutomation/base" defines the shortcut, which can or has to be used for global objects. The shortcut **mcrc** corresponds to the abbreviation of “Postal Automation MC/RC Base”.

The shortcut can be used in the schema as shown in the example:

```
<complexType name="QuadPolygonType">
    <sequence minOccurs="4" maxOccurs="4">
        <element name="dot" type="mcrc:DotType"/>
    </sequence>
</complexType>
```

DotType is a global type, which is assigned to the element “dot”. Due to the fact that the type is defined globally, it has to have the namespace shortcut prefix. The extension is required to enable applications to locate the global type.

6.2.3 Domain Data Model & Types

The following general introduction to the data model is taken without changes from CEN/TS 15448. It is repeated here for the reader’s convenience.

The data model can be divided in several sub sections, which represent different abstraction levels.

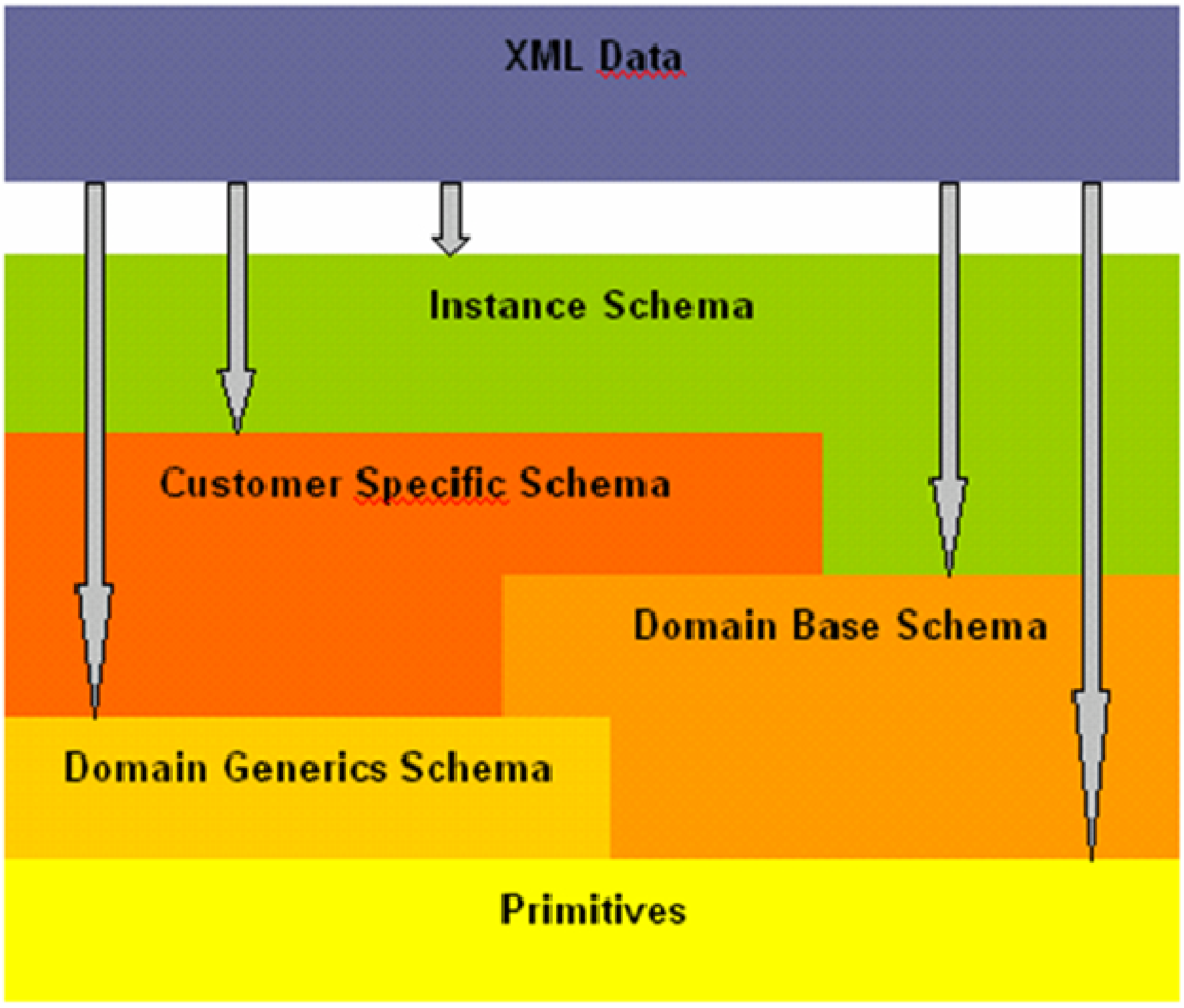


Figure 20 — Data Model Composition (Layer View)

a. **Domain Primitives:**

The lowest level contains primitive data types. These primitive types are used in all higher abstraction levels. The layer contains derived and restricted simple XML build-in data types with poor abstraction level. There is no distinction in the primitive’s layer between common and postal domain specific types to avoid higher fragmentation.

b. Domain Generic Types:

The generic section provides all type specifications used in the base and customer specification and forms the backbone of the domain type system. Both higher layers tailor the generic types in some fashion. The data types are explicitly postal domain data types.

c. Domain Base Types:

The domain base layer specifies types, which are usually valid within the overall domain. These types need no project specific customization concerning the static structure of the composed data types – e.g. batch coordinates as a rule contain no customer specific information like the distribution code in a recipient address result.

To avoid misunderstanding; the static structure of a base type is fix, the content must possibly be adapted due to the customers requirements – e.g. a barcode result typically provides information about the symbology of the specific barcode. The symbology – e.g. 4-state code, postnet code - is customer specific.

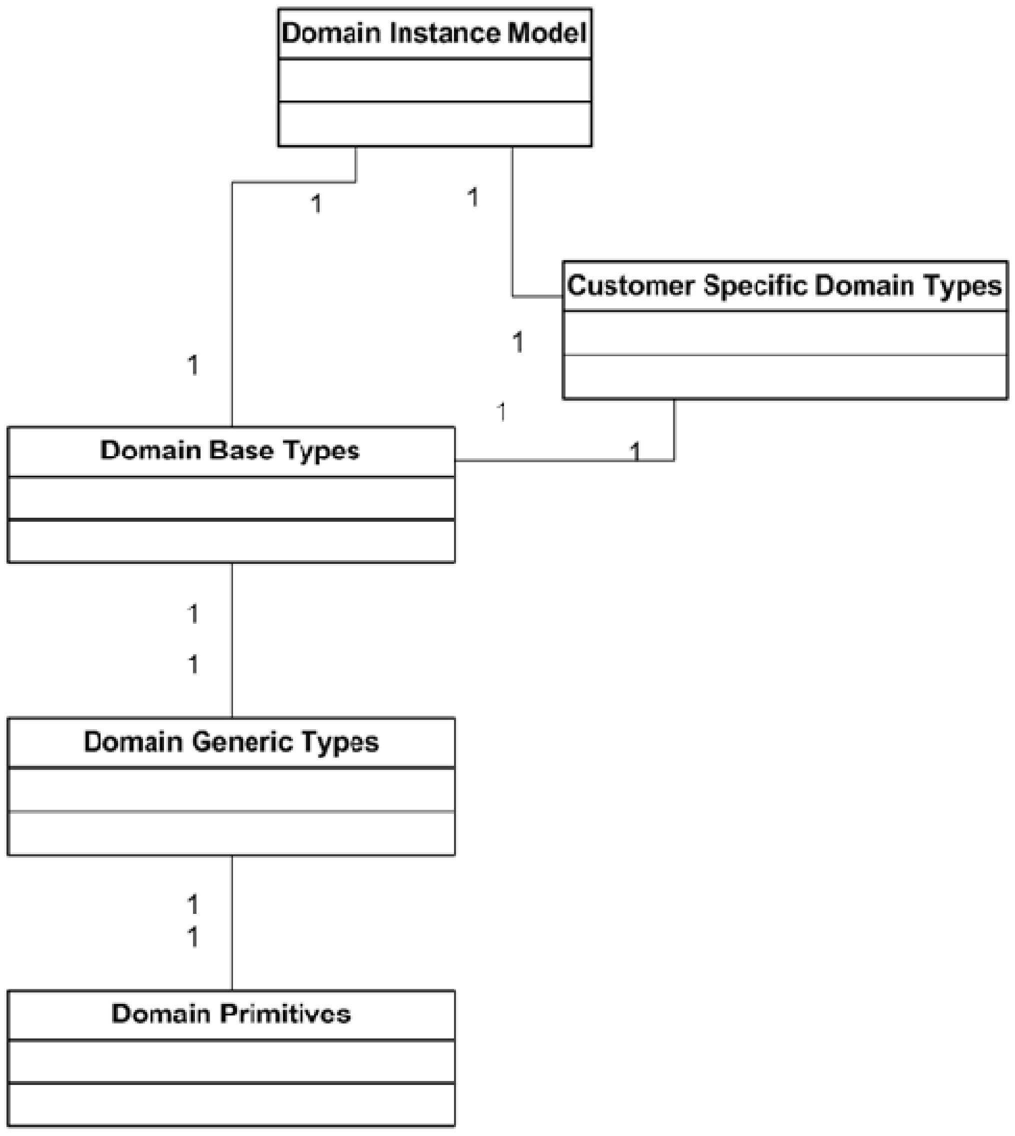


Figure 21 — Data Model Composition (UML)

d. **Customer Specific Domain Types:**

The customer specific types inherit both, the base types and the generic types. The base types have to be adapted if a customer specific adaptation is required (should be the exception). The generic types have to be adapted to gain the full advantage of XML schemas and capability of instance document validation. The domain specific adaptations tailor the static structure of a generic type and add customer semantics – e.g. the specification for the distribution code syntax.

e. Domain Instance Model:

The domain instance model represents the top level layer. It includes the base type layer and the customer specific layer.

NOTE It should be the goal, to standardize as much as possible. This applies to the static type structure as well as to the semantics. All standard specifications should be located in the domain base part. The reduction of the customer specific specifications reduces engineering efforts significantly.

The following subclauses cover the components of the data model.

6.2.3.1 Domain Instance Types

The domain instance section provides the types for the xml messages which are exchanged in the Use Cases defined for this interface. There is a common root element for all messages. This element is named MC_RC_Message and shown in Figure 22.

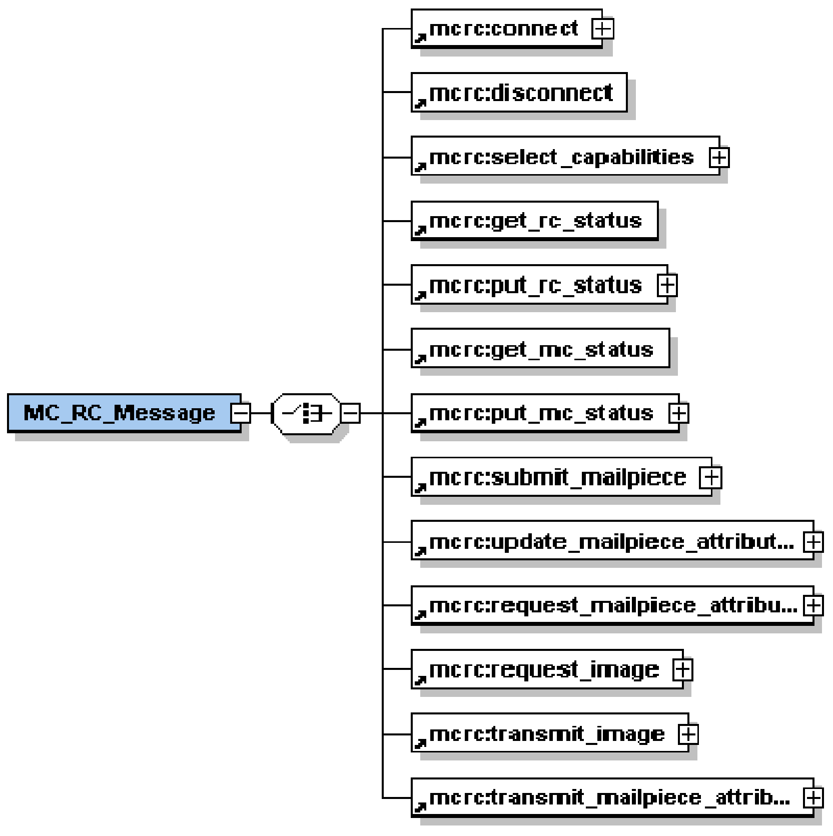


Figure 22 — Schema of MC_RC_Message

The child elements of the element MC_RC_Message are briefly described and mapped to use cases in the following table:

Table 41 — Mapping between Messages, Domain Instance Types, and Use Cases

Domain Instance Type	Brief Description	Transmitted in Use Case
connect	This element allows the MC to pass information to the RC regarding the connection itself.	Connect
disconnect	This element does not contain any data.	Disconnect
select_capabilities	This element restricts the capabilities that the MC uses from the RC.	Connect
get_rc_status	This element does not contain any data.	Get RC System Status
put_rc_status	This element allows the RC to provide state information to the MC.	Put RC System Status
get_mc_status	This element does not contain any data.	Get Machine Status
put_mc_status	This element allows the MC to provide state information to the RC.	Put Machine Status
submit_mailpiece	This element specifies the content of a request sent from the MC to the RC System.	Submit Mailpiece
update_mailpiece_attributes	This element allows the update of a mail piece’s attributes stored at the RC.	Update Mailpiece Attributes
request_mailpiece_attributes	This element specifies a mailpiece for which the MC requests the available attributes from the RC.	Request Mailpiece Attributes
request_image	This element specifies specific attributes related to an image request from the MC to the RC	Request Image
transmit_image	This element can specify attributes of an image transmitted from the RC to the MC.	Transmit Image
transmit_mailpiece_attributes	This element specifies the content of a response returned from the RC regarding a specific mail piece.	Transmit Mailpiece Attributes

The following subclauses describe the mentioned types in detail.

6.2.3.1.1 Type connect

This type allows the MC to pass information to the RC regarding the connection itself.

```
<element name="connect">
  <complexType>
    <sequence>
      <element name="interface_version_id" type="string"/>
      <element name="machine_controller_id" type="string"/>
      <element name="feeder_id" type="string"/>
      <element name="heartbeat_time_int" type="unsignedInt"/>
      <element name="initial_batch_information" type="mcrc:BatchInformationT" minOccurs="0"/>
    </sequence>
  </complexType>
</element>
```

Table 42

Element	Description
interface_version_id	The element **interface_version_id** is a string representing the current version identifier for the MC-RC interface.
machine_controller_id	The element **machine_controller_id** provides an identifier indicating which MC is connecting to the RC System.
feeder_id	The element **feeder_id** provides an identifier indicating which feeder of the MC is connecting to the RC System.
heartbeat_time_int	The element **heartbeat_time_int** provides the desired time, in milliseconds, for the MC and RC System to exchange regular heartbeat functions.
initial_batch_information	The optional element **initial_batch_information** provides the batch coordinates and related processing configuration that shall be used for the connection. For the definition of the corresponding type BatchInformationT see 6.2.3.2.13.

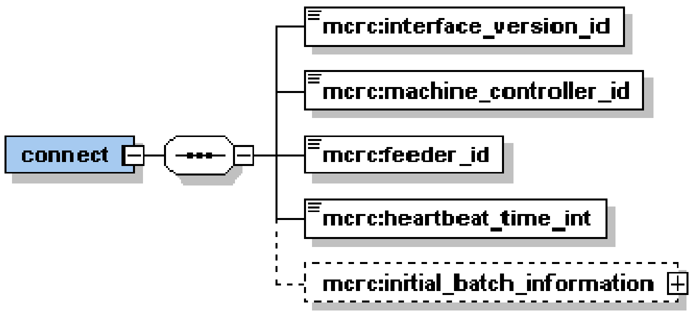

connect
mcrc:interface_version_id
mcrc:machine_controller_id
mcrc:feeder_id
mcrc:heartbeat_time_int
mcrc:initial_batch_information

Figure 23 — XML representation of connect

6.2.3.1.2 Type select_capabilities

This type specifies the capabilities that the MC selects from the RC.

```
<element name="select_capabilities" type="mcrc:CapabilitySetT"/>
```

Table 43

Element	Description
CapabilitySetT	A set of capabilities. See definition in 6.2.3.2.4

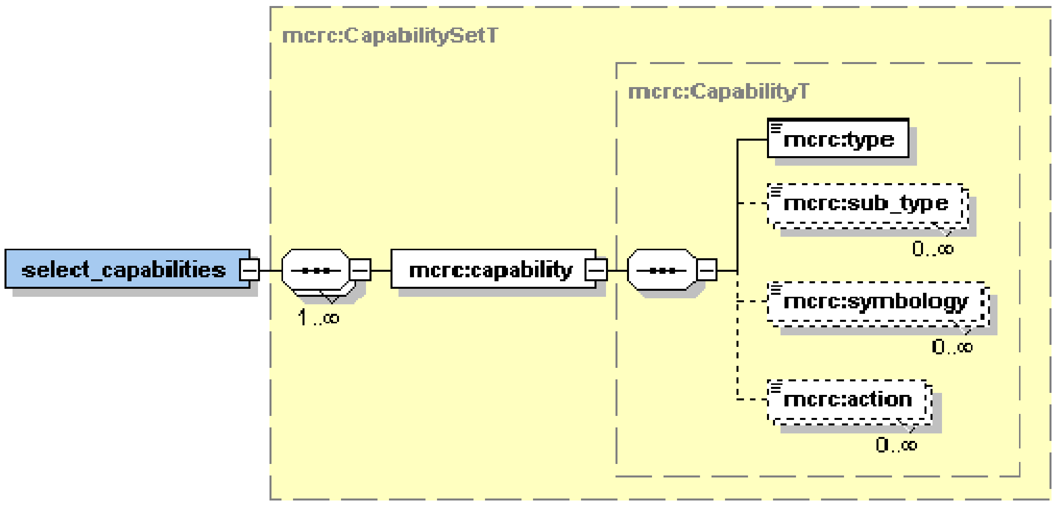


Figure 24 — XML representation of select_capabilities

6.2.3.1.3 Type get_rc_status

This empty type does not contain any data. It might be extended in later versions of this interface or in project specific incarnations.

```
<element name="get_rc_status"/>
```

6.2.3.1.4 Type put_rc_status

This type serves as a data structure for the RC System to provide status to the MC.

```
<element name="put_rc_status">
  <complexType>
    <sequence>
      <element name="state" type="mcrc:RCStateT"/>
      <element name="text" type="string" minOccurs="0"/>
      <element name="selected_ready_capabilities" type="mcrc:CapabilitySetT"/>
      <element name="selected_not_ready_capabilities" type="mcrc:CapabilitySetT"/>
    </sequence>
  </complexType>
</element>
```

Table 44

Element	Description

State	The element **state** refers to the state of the RC System at the current time.
	
	This value is an enumerated list:
	
	`<simpleType name="RCStateT">`
	`    <restriction base="NMTOKEN">`
	`        <enumeration value="ready"/>`
	`        <enumeration value="not_ready"/>`
	`        <enumeration value="initializing"/>`
	`    </restriction>`
	`</simpleType>`
	
	Use “ready” when at least one selected capability is ready.
	Use “not_ready” when a central component of the RC System is not available or when none of the selected capabilities are ready.
	Use “initializing” when the RC System is connected but not yet fully started.

Text	The element **text** provides a variable-length text buffer the RC System may use to provide a status description for logging purposes
selected_ready_capabilities	This element contains the selected capabilities that are currently ready at the RC System. Before selection of specific capabilities by the MC this set contains all the ready capabilities of the RC. For the definition of CapabilitySetT see 6.2.3.2.4
selected_not_ready_capabilities	This element contains the selected capabilities that are currently not ready at the RC System. Before selection of specific capabilities by the MC this set contains all the not ready capabilities of the RC. For the definition of CapabilitySetT see 6.2.3.2.4

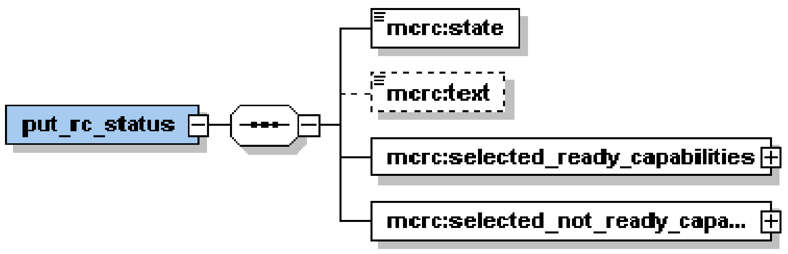

put_rc_status
mcrc:state
mcrc:text
mcrc:selected_ready_capabilities
mcrc:selected_not_ready_capa...

Figure 25 — XML representation of put_rc_status

6.2.3.1.5 Type get_mc_status

This empty type does not contain any data. It might be extended in later versions of this interface or in project specific incarnations.

```
<element name="get_mc_status"/>
```

6.2.3.1.6 Type put_mc_status

This type serves as a data structure for the MC to provide status to the RC System.

```
<element name="put_mc_status">
    <complexType>
        <sequence>
          <element name="state" type="mcrc:MCStateT"/>
          <element name="text" type="string" minOccurs="0"/>
        </sequence>
    </complexType>
</element>
```

Table 45

Element	Description

State	The element **state** refers to the state of the MC at the current time.
	
	This value is an enumerated list:
	
	`<simpleType name="MCStateT">`
	    `<annotation>`
	        `<documentation>`
	        `- machine is running`
	        `- machine is stopped`
	        `- machine is in test mode`
	        `- machine is in maintenance mode`
	        `</documentation>`
	    `</annotation>`
	    `<restriction base="NMTOKEN">`
	        `<enumeration value="running"/>`
	        `<enumeration value="stopped"/>`
	        `<enumeration value="test"/>`
	        `<enumeration value="maintenance"/>`
	    `</restriction>`
	`</simpleType>`

Text	The element **text** provides a variable-length text buffer the MC may use to provide a status description for logging purposes

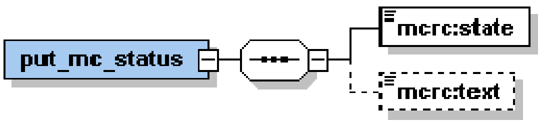


Figure 26 — XML representation of put_mc_status

6.2.3.1.7 Type submit_mailpiece

The type submit_mailpiece provides the description of the content of a processing request. The MC creates the request and supplies the RC with the document. The complete listing of the schema specification is located in the appendix in A.3.

```
<element name="submit_mailpiece">
    <complexType>
        <complexContent>
            <extension base="mcrc:MailpieceSubmissionT">
                <attributeGroup ref=" mcrc:MCRCInfrastructureDataT"/>
                <attributeGroup ref=" mcrc:ProcessingConfigurationT"/>
                <attributeGroup ref=" mcrc:ProcessingInstructionsT"/>
            </extension>
        </complexContent>
    </complexType>
</element>
```

Table 46

Element	Description
MailpieceSubmissionT	Basic element, which holds the data for a recognition request. For the definition of MailpieceSubmissionT see 0

MCRCInfrastructureDataT	The group holds the basic and unique identifier used and required for identification and tracking purposes. This data structure is defined uniquely from that available in the IC-ED interface, as it was determined that **submission_id** is not required at this level.
	
	The **mailpiece_id** represents the identifier for the mail item, which has to be processed by the RC.
	
	The **id_tag** represents a long living identifier for the mail item.
	
	`<attributeGroup name="MCRCInfrastructureDataT">`
	`    <attribute name="mailpiece_id" type="string" use="required"/>`
	`    <attribute name="id_tag" type="string" use="optional"/>`
	`</attributeGroup>`

ProcessingConfigurationT	ProcessingConfigurationT provides information about the overall process environment. **source_id** identifies the sorting machine. **feeder_id** refers to the feeding unit, where the mailpiece image was lifted. The attribute **sortplan_id** provides the id of the sortplan selected from a machine operator. The **lift_location** specifies the sorting centre where the image lift occurred. The **geo_mode** provides information about the geographic destination of the mail batch (see 6.2.3.4.3). The **mail_flow** attribute specifies the mail flow currently processed. The values can be *outgoing* : mail is to be delivered all around the country (and outbound) *regional* : mail is to be delivered in the region of the sorting centre *incoming* : mail is to be delivered within the sorting centre area

	`<attributeGroup name="ProcessingConfigurationT">`
	`<attribute name="source_id" type="string" use="optional"/>`
	`<attribute name="feeder_id" type="string" use="optional"/>`
	`<attribute name="sortplan_id" type="string" use="optional"/>`
	`<attribute name="lift_location" type="string" use="optional"/>`
	`<attribute name="geo_mode" type="BatchCntryCodeAttrT" use="optional"/>`
	`<attribute name="mail_flow" type="string" use="optional"/>`
	`</attributeGroup>`

ProcessingInstructionsT	The group contains all elements which instruct or control the recognition process.
	
	permitted_time specifies the allowed processing time for the individual submission. It is set by the MC. The unit of **permitted_time** is milliseconds (ms).
	
	`<attributeGroup name="ProcessingInstructionsT">`
	`<attribute name="permitted_time" type="unsignedInt" use="optional"/>`
	`</attributeGroup>`

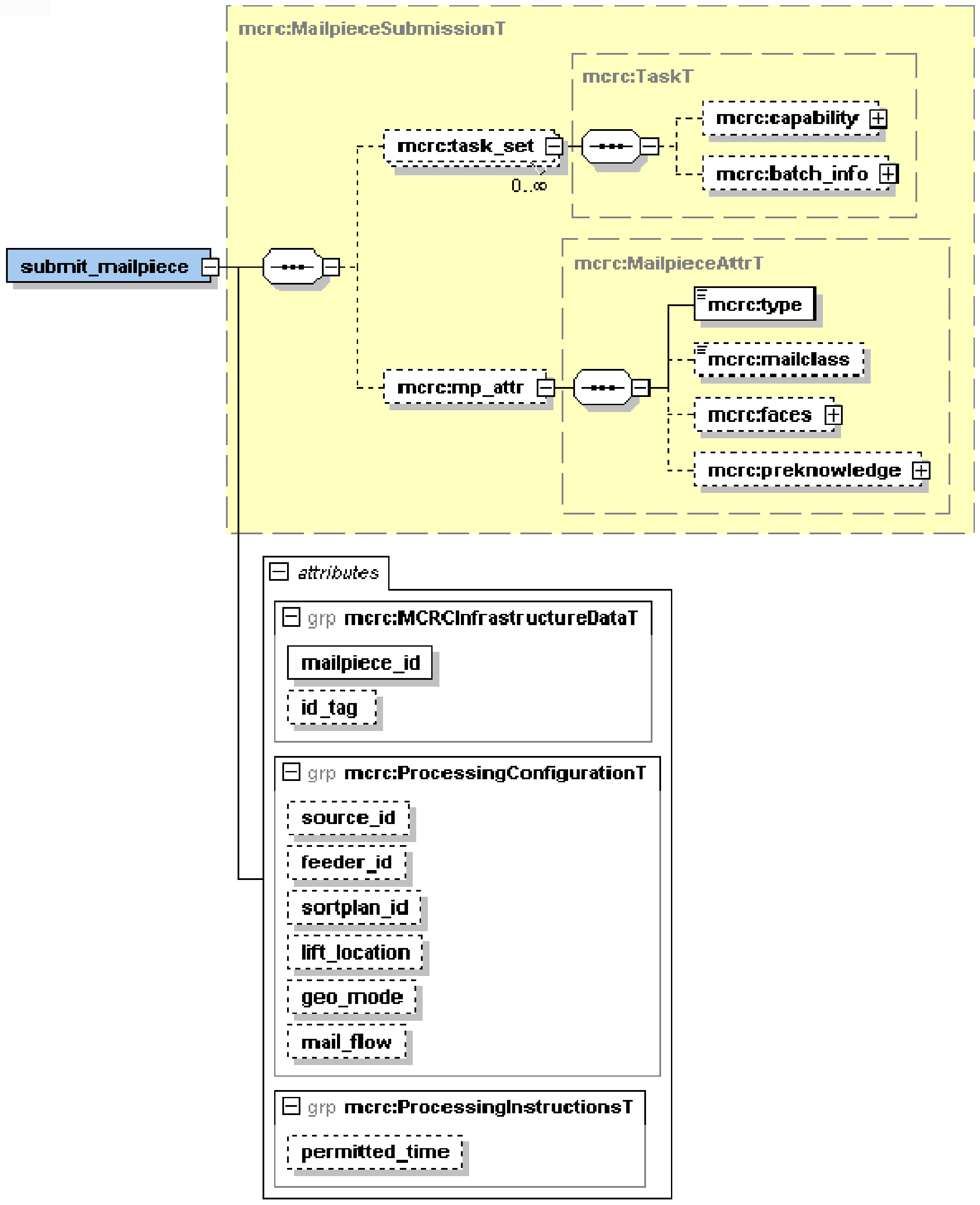


Figure 27 — XML representation of submit_mailpiece

6.2.3.1.8 Type update_mailpiece_attributes

The Update Mailpiece Attribute Type allows for attribute updates.

```
<element name="update_mailpiece_attributes">
      <complexType>
            <sequence>
                  <element name="iood" type="boolean" minOccurs="0"/>
                  <element name="weight" type="unsignedInt" minOccurs="0"/>
                  <element name="indicia_value" type="string" minOccurs="0"/>
```

```
                <element name="indicia_type" type="string" minOccurs="0"/>
                <element name="sort_bin" type="unsignedInt" minOccurs="0"/>
                <element name="idtag_print_status" type="boolean" minOccurs="0"/>
                <element name="code" type="string" minOccurs="0"/>
                <element name="final_coded" type="boolean" minOccurs="0"/>
            </sequence>
            <attribute name="mailpiece_id" type="string" use="required"/>
            <attribute name="source_id" type="string" use="optional"/>
            <attribute name="scanner_id" type="string" use="optional"/>
            <attribute name="feeder_id" type="string" use="optional"/>
            <attribute name="sortplan_id" type="string" use="optional"/>
        </complexType>
    </element>
```

Table 47

Element	**Description**
iood	A boolean element indicating whether or not the mail piece is passing outside the mail processing machine's delay line.
weight	Weight, in grams, of the mail piece.
indicia_value	The value of the indicia printed on the mail piece. Specific values are defined on a per-program basis.
indicia_type	The type of the indicia printed on the mail piece. Specific values are defined on a per-program basis.
sort_bin	The sort bin to which the machine physically sorted the mail piece.
idtag_print_status	A boolean value indicating whether or not an ID-Tag was printed on the mail piece.
code	The destination sort code assigned to the mail piece. Specific values are defined on a per-program basis.
final_coded	A boolean value indicating whether or not processing for a mail piece is complete.
Attribute	**Description**
mailpiece_id	This attribute specifies the id of the mail piece.
source_id	This attribute specifies the id of the sorting machine processing the mail piece.
scanner_id	This attribute specifies the id of the scanner where the mail piece was scanned.
feeder_id	This attribute specifies the id of the feeder where the mail piece was lifted.
sortplan_id	This attribute specifies the id of the sortplan.

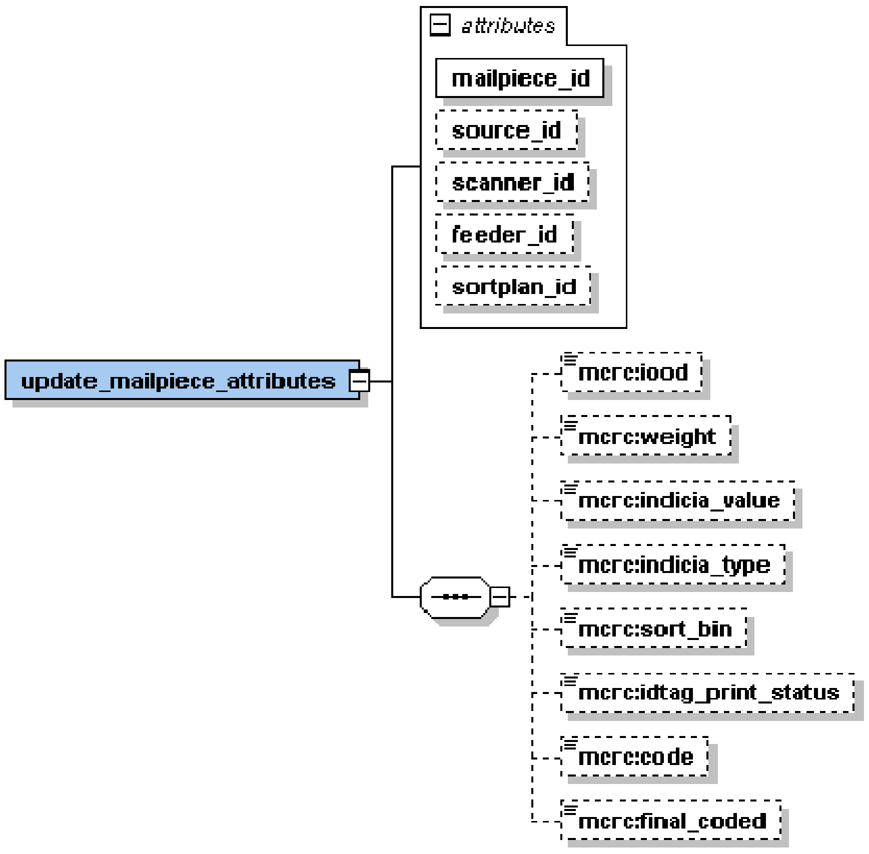


Figure 28 — XML representation of update_mailpiece_attributes

6.2.3.1.9 Type request_mailpiece_attributes

This type provides the requester with the ability to obtain the results of an enrichment response. The RC returns the document either on request of the MC or in an unsolicited fashion.

The MC expects to get all defined attributes for the corresponding mailpiece from the RC System.

```
<element name="request_mailpiece_attributes">
  <complexType>
          <attributeGroup ref="mcrc:MCRCInfrastructureDataT"/>
          <attributeGroup ref="mcrc:ProcessingInstructionsT"/>
  </complexType>
</element>
```

Table 48

Element	Description
MCRCInfrastructureDataT	This type holds the attribute **mailpiece_id** which is the identifier for the mail item.
ProcessingInstructionsT	This type holds the attribute **permitted_time** which specifies the time in milliseconds in which the MC expects a response from the RC System.

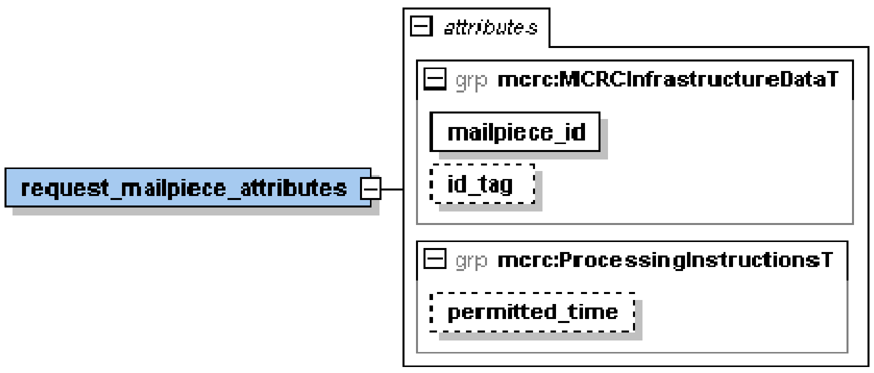

attributes
grp mcrc:MCRCInfrastructureDataT
mailpiece_id
id_tag
grp mcrc:ProcessingInstructionsT
permitted_time
request_mailpiece_attributes

Figure 29 — XML representation of request_mailpiece_attributes

6.2.3.1.10 Type request_image

This type allows the specification of the image requested.

```
<element name="request_image" type="mcrc:ImageAttributesT"/>
```

Table 49

Element	Description
ImageAttributesT	This type holds the attributes of the requested image. For the definition of ImageAttributeT see 6.2.3.2.5

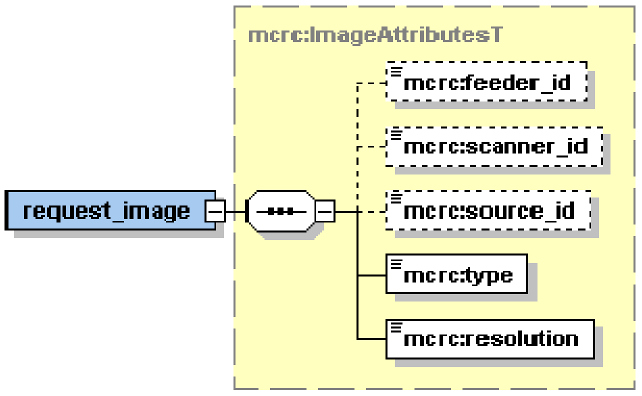


Figure 30 — XML representation of request_image

6.2.3.1.11 Type transmit_image

This type allows the specification of the image transmitted.

Mind, that there might be multiple images transmitted for the same mail piece using a set of TIFF containers (one per image) or a single multi-page TIFF container.

```
<element name="transmit_image">
     <complexType>
          <sequence>
            <element name="request_parameters" type="mcrc:ImageAttributesT" minOccurs="0"/>
      <element name="image_status" type="mcrc:ImageStatusT" minOccurs="0"/>
            <element name="available_attributes" type="mcrc:RCResultT" minOccurs="0"/>
          </sequence>
     </complexType>
</element>
```

Table 50

Element	Description
request_parameters	This element of type ImageAttributesT holds the attributes of the original request for the transmitted image. For the definition of ImageAttributesT see 6.2.3.2.5.
image_status	This element of enumeration type ImageStatusT contains the status of the requested image. Possible values include: • **ok** means that image is available • **no_image** means that no image with requested parameters is available • **invalid_id** means that id of e.g feeder or scanner is unknown for the Machine
available_attributes	This element holds the attributes that are available for the corresponding mailpiece. For the definition of RCResultT see 6.2.3.2.1.

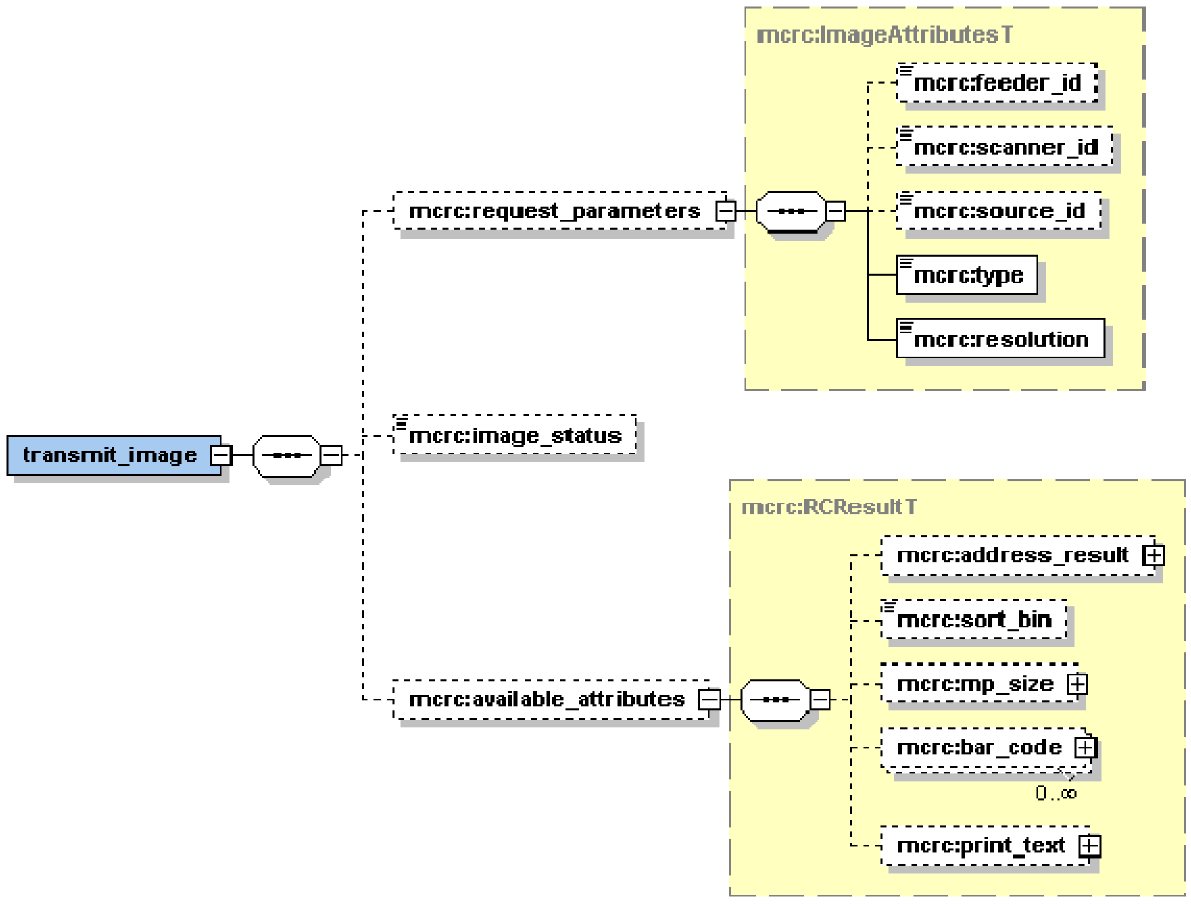


Figure 31 — XML representation of transmit_image

6.2.3.1.12 Type transmit_mailpiece_attributes

This type contains the result for a mail piece that has been determined by the RC System.

```
<element name="transmit_mailpiece_attributes">
  <complexType>
        <complexContent>
          <extension base="mcrc:RCResultT">
              <attributeGroup ref="mcrc:MCRCInfrastructureDataT"/>
              <attributeGroup ref="mcrc:ProcessingInstructionsT"/>
          </extension>
        </complexContent>
  </complexType>
</element>
```

Table 51

Element	Description
RCResultT	Basic Element which holds the result for a mail piece. For the definition of RCResultT see 6.2.3.2.1
MCRCInfrastructureDataT	This type holds the attribute • **mailpiece_id** which is the identifier for the mail item. • **id_tag** which is a long living identifier for the mail item.
ProcessingInstructionsT	This type holds the attribute **permitted_time** which specifies the time in milliseconds in which the MC expects a response from the RC System.

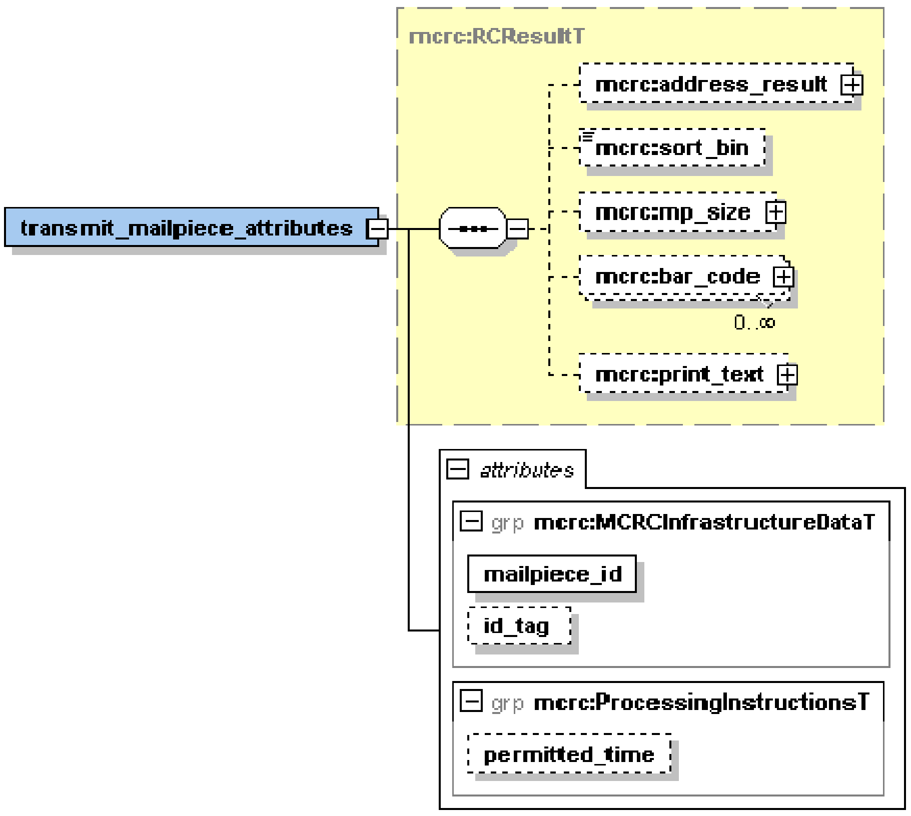


Figure 32 — XML representation of transmit_mailpiece_attributes

6.2.3.2 Domain Generic Types

6.2.3.2.1 RC Result Type

This type contains the results that the RC System sends to the MC. It is used by the instance types *transmit_mailpiece_attributes* and *transmit_image*.

```
<complexType name="RCResultT">
    <sequence>
        <element name="address_result" type="mcrc:AddressResultT" minOccurs="0"/>
        <element name="sort_bin" type="string" minOccurs="0"/>
        <element name="mp_size" type="mcrc:DimensionT" minOccurs="0"/>
        <element name="bar_code" type="mcrc:BarCodeT" minOccurs="0" maxOccurs="unbounded"/>
        <element name="print_text" type="mcrc:PrintTextT" minOccurs="0"/>
    </sequence>
</complexType>
```

Table 52

Element	Description
address_result	This element contains the results related to address recognition. For the definition of AddressResultT see 6.2.3.2.2
sort_bin	The sort bin, if one has been assigned by the RC System.
mp_size	This element contains the size of the (physical) mail piece. If the RC System does not measure or know the size of the mail piece, this element is absent. For the type DimensionT see 6.2.3.2.7.
bar_code	If the RC System detects a bar code on the mail piece it can use this element to send the bar code to the MC. For the type BarCodeT see 6.2.3.2.8.
print_text	This element contains text to be printed on the mailpiece. For the definition of PrintTextT see 6.2.3.2.3.

6.2.3.2.2 Address Result Type

This element contains the results related to address recognition.

```
<complexType name="AddressResultT">
    <sequence>
        <element name="code" type="string" minOccurs="0"/>
        <element name="code_attribute" type="string" minOccurs="0" maxOccurs="unbounded"/>
        <element name="final_coded" type="boolean" minOccurs="0"/>
        <element name="status_attr" type="string" minOccurs="0"/>
        <element name="result_source" type="string" minOccurs="0"/>
        <element name="reject_reason" type="string" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

Table 53

Element	Description
code	The distribution code determined by the RC System. Unless a reject reason is provided, this element must be present. This element needs to be consistent with the OCR/VCS standard.
code_attribute	This element contains information about the depth of the determined result code. The values have to be consistent with the AddressCodeAttrT in the OCR/VCS standard. If the element “code” is absent, this element is also absent. Possible example values include: • “str”, means Street • “pob”, means Post Office Box
status_attr	This element contains the result status. The value has to be consistent with the AddressStatusAttrT of the OCR/VCS standard. Possible values include: • “fin”, means result is finalized, no further work is required. • “fdos”, means result code is of finest depth of sort.
result_source	This string element specifies where the result was obtained. It could correspond to the value of the attribute “creator” in the type RecognitionResultT type of the OCR/VCS standard. E.g., it could be the name of the OCR identifying the result or the ID of a VCD keyer.
reject_reason	This string element contains a reason why the RC System was unable to determine a result.

6.2.3.2.3 Print Text Type

This element contains text that has to be printed on a mail piece.

```
<complexType name="PrintTextT">
    <sequence>
        <element name="type" type="string"/>
        <element name="text_line" type="string" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

Table 54

Element	Description
type	The type of information that the text contains, e.g. “forwarding”.
text_line	A line of text to be printed on the mailpiece.

6.2.3.2.4 Capability Set Type

The Capability Set type specifies the content of a XML structure which provides information about the capabilities of an RC.

```
<element name=CapabilitySetT">
    <complexType>
        <sequence maxOccurs="unbounded">
            <element name="capability" type="mcrc:CapabilityT"/>
        </sequence>
    </complexType>
</element>
```

Table 55

Element	Description
capability	Basic element which allows specifying a single RC capability or a set of capabilities. The **CapabilityT** type is assigned to the **capability** element. The detailed description of **CapabilityT** can be found in 6.2.3.2.11.

6.2.3.2.5 Image Attributes Type

The Image Attributes type serves as a data structure accompanying the request for an image stored by the RC System. This structure provides data sufficient to allow the RC to select an image appropriate to the type desired by the MC.

```
<complexType name="ImageAttributesT">
    <sequence>
        <element name="feeder_id" type="string" minOccurs="0"/>
        <element name="scanner_id" type="string" minOccurs="0"/>
        <element name="source_id" type="string" minOccurs="0"/>
        <element name="type" type="mcrc:ImageTypeT"/>
        <element name="resolution" type="mcrc:ImageResolutionT"/>
    </sequence>
</complexType>
```

Table 56

Element	Description
feeder_id	The **feeder_id** specifies from which feeder the image shall originate.
scanner_id	The **scanner_id** specifies from which scanner the image shall originate.
source_id	This element specifies the id of the sorting machine processing the mail piece.

type	The type of the image, where “all” means all available types.
	`<simpleType name="ImageTypeT">`
	    `<restriction base="NMTOKEN">`
	        `<enumeration value="all"/>`
	        `<enumeration value="binary"/>`
	        `<enumeration value="grey"/>`
	        `<enumeration value="color"/>`
	    `</restriction>`
	`</simpleType>`

resolution	The **resolution** of the image, where “all” means all available resolutions.
	`<simpleType name="ImageResolutionT">`
	    `<restriction base="NMTOKEN">`
	        `<enumeration value="all"/>`

	`<enumeration value="high"/>`
	`<enumeration value="low"/>`
	`</restriction>`
	`</simpleType>`

6.2.3.2.6 RC State Information Type

The RC State Information type serves as a data structure for the RC System to provide status to the MC

```
<simpleType name="RCStateT">
      <restriction base="NMTOKEN">
            <enumeration value="ready"/>
            <enumeration value="notready"/>
            <enumeration value="initializing"/>
            <enumeration value="notconnected"/>
      </restriction>
</simpleType>
```

Table 57

Element	Description

state	**state** refers to the state of the RC System at the current time.
	
	This value is an enumerated list:
	
	`<simpleType name="RCStateT">`
	    `<restriction base="NMTOKEN">`
	        `<enumeration value="ready"/>`
	        `<enumeration value="not_ready"/>`
	        `<enumeration value="initializing"/>`
	        `<enumeration value="notconnected"/>`
	    `</restriction>`
	`</simpleType>`
	
	Use “initializing” when the RC System is connected but not yet configured (in terms of setting capabilities).

text	**text** provides a variable-length text buffer the RC System may use to provide a status description for logging purposes

6.2.3.2.7 Dimension Type

This type contains the mailpiece’s dimensions in millimetres.

```
<complexType name="DimensionT">
    <sequence>
        <element name="long" type="unsignedInt" minOccurs="0"/>
        <element name="high" type="unsignedInt" minOccurs="0"/>
        <element name="thick" type="float" minOccurs="0"/>
    </sequence>
</complexType>
```

Table 58

Element	Description
long	This element contains the physical length of the mail item in millimetres.
high	This element contains the physical height of the mail item in millimetres.
thick	This element contains the physical thickness of the mail item in millimetres. Mind, that this value is of type float.

6.2.3.2.8 Barcode Type

This type contains a barcode.

```
<complexType name="BarCodeT">
    <sequence>
        <element name="type" type="mcrc:BarcodeSymbologyT" minOccurs="0"/>
        <element name="value" type="string" minOccurs="0"/>
        <element name="status" type="mcrc:BarCodeStatusT"/>
    </sequence>
</complexType>
```

Table 59

Element	Description

type	BarcodeSymbologyT provides the valid symbologies of a barcode label.
	
	`<simpleType name="BarcodeSymbologyT">`
	  `<restriction base="NMTOKEN">`
	    `<enumeration value="PostNet"/>`
	    `<enumeration value="Planet"/>`
	    `<enumeration value="RM4SCC"/>`
	    `<enumeration value="USPS4SC"/>`
	    `<enumeration value="KIXC"/>`
	    `<enumeration value="AUPC"/>`
	    `<enumeration value="2of5I"/>`
	    `<enumeration value="I2of5"/>`
	    `<enumeration value="Codabar"/>`
	    `<enumeration value="AIM39"/>`
	    `<enumeration value="AIM93"/>`
	    `<enumeration value="AIM128"/>`
	    `<enumeration value="UPC-A"/>`
	    `<enumeration value="UPC-E"/>`
	    `<enumeration value="EAN-8"/>`
	    `<enumeration value="EAN-13"/>`
	    `<enumeration value="EAN-128"/>`
	    `<enumeration value="RSS-14"/>`
	    `<enumeration value="PDF417"/>`
	    `<enumeration value="Aztec"/>`
	    `<enumeration value="DataMatrix"/>`
	    `<enumeration value="MaxiCode"/>`
	    `<enumeration value="FIM"/>`
	  `</restriction>`
	  `<!--`
	    Postnet - USPS Postnet barcode
	    Planet - USPS Planet barcode
	    RM4SCC - Royal Mail 4-State Customer Code
	    USPS4SC - USPS 4-State Code
	    KIXC - Netherlands 4-State Customer Code
	    AUPC - Australian Postal Code - 4-State Barcode
	    2of5I - 2 of 5 Code - Industrial
	    I2of5 - Interleaved 2 of 5 Code
	    Codabar
	    AIM39 - Code 39
	    AIM93 - Code 93
	    AIM128 - Code 128
	    UPC-A
	    UPC-E - short version of UPC-A
	    EAN-8 - European Article Number - Length 8
	    EAN-13
	    EAN-128
	    RSS-14 - Reduced Space Symbology
	    PDF417 - Potable Data File 417
	    Aztec
	    DataMatrix
	    MaxiCode
	    FIM
	  `-->`
	`</simpleType>`

value	This element contains the value of the barcode.

status	This element contains the status of the barcode. Valid status are defined by the values of the enumeration BarCodeStatusT.
	
	`<simpleType name="BarCodeStatusT">`
	`    <restriction base="NMTOKEN">`
	`        <enumeration value="no_barcode"/>`
	`        <enumeration value="detected_and_recognized"/>`
	`        <enumeration value="detected_and_not_recognized"/>`
	`    </restriction>`
	`</simpleType>`

6.2.3.2.9 Mailpiece Submission Type

MailpieceSubmissionT is a complex type and integrates the different request elements.

```
<complexType name="MailpieceSubmissionT">
    <sequence>
        <element name="task_set" type="mcrc:TaskT" minOccurs="0" maxOccurs="unbounded"/>
        <element name="mp_attr" type="mcrc:MailpieceAttrT" minOccurs="0"/>
    </sequence>
</complexType>
```

Table 60

Element	**Description**
task_set	The **task_set** element represents a container and stores capability specific information in single data sets. An MC can support the RC with multiple “task sets” and as a result of that request capability specific recognition. For detailed information to the TaskT type refer to 6.2.3.2.10 When this element is missing the RC System is expected to apply all selected capabilities.

mp_attr	The MailpieceAttrT type is assigned to the element **mp_attr**. It provides basic mailpiece related information:
	- the kind of the mail item
	- the mailclass of the mail item; important for the ED because it and may affect the address recognition
	- the scanned **faces** of the mail item. The FaceSetT type is assigned to **faces** and allows providing a sequence of faces. The FaceSetT types allows to link a face with the corresponding image by a page attribute.
	- a preknowledge result if a barcode reader or some other type of Enrichment Device is directly attached to the MC. This result will be in accordance with the result type specified in the IC-ED interface.
	
	`<complexType name="MailpieceAttrT">`
	`  <sequence>`
	`    <element name="type" type="mcrc:MailpieceTypeT"/>`
	`    <element name="mailclass" type="string" minOccurs="0"/>`
	`    <element name="faces" type="mcrc:FaceSetT" minOccurs="0"/>`
	`    <element name="preknowledge" type="mcrc:RecognitionResultT"`
	`                                    minOccurs="0"/>`
	`  </sequence>`
	`</complexType>`
	
	The detailed specification of MailpieceAttrT can be found in 6.2.3.5.2

6.2.3.2.10 Task Type

TaskT provides all information which RC requires to process the requested task(s).

```
<complexType name="TaskT">
    <sequence>
        <element name="capability" type="mcrc:CapabilityT" minOccurs="0"/>
        <element name="batch_info" type="mcrc:BatchDescriptionT"  minOccurs="0"/>
    </sequence>
</complexType>
```

Table 61

Element	Description
capability	The **capability** element holds the description of the tasks which have to be processed by the RC. The description and structure is strongly dependent on the specific capability.
batch_info	The **batch_info** element allows the MC to specify batch characteristics to apply to the mailpiece.

6.2.3.2.11 Capability Type

The **capability** enables an MC to specify a detailed recognition request and to control the recognition of an RC. It further puts an RC in the position to forward its capabilities to an MC.

In difference to the CapabilityT used in the OCR/VCS interface, this type is extended with the values “online” and “offline” in the ActionT.

```
<complexType name="CapabilityT">
    <sequence>
        <element name="type" type="mcrc:CapabilityTypeT"/>
        <element name="sub_type" type="string" minOccurs="0" maxOccurs="unbounded"/>
        <element name="symbology" type="string" minOccurs="0" maxOccurs="unbounded"/>
        <element name="action" type="mcrc:ActionT" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

Table 62

Element	Description

Type	The **type** element is linked with the simple type CapabilityTypeT. It specifies the valid scale of values of RC capabilities.
	
	`<simpleType name="CapabilityTypeT">`
	`    <restriction base="NMTOKEN">`
	`        <enumeration value="rec_code"/>`
	`        <enumeration value="sen_code "/>`
	`        <enumeration value="mcl_code "/>`
	`        <enumeration value="end_code "/>`
	`        <enumeration value="dbc_code "/>`
	`        <enumeration value="cbc_code "/>`
	`        <enumeration value="pbc_code "/>`
	`        <enumeration value="result_management "/>`
	`        <enumeration value="image_request"/>`
	`        <enumeration value="synch_result_trans "/>`
	`        <enumeration value="asynch_result_trans"/>`
	`    </restriction>`
	`</simpleType>`
	
	rec_code – recipient address recognition
	sen_code – sender address recognition
	mcl_code – mailclass determination
	end_code – endorsement line recognition
	dbc_code – distribution barcode recognition
	cbc_code – customer barcode recognition
	pbc_code – product barcode recognition
	
	result_management – Storage/Query of enrichment results e.g. by ID Tag

image_request – request image by MC, e.g. for selection of batch region

synch_result_trans – Synchronous result transmission

asynch_result_trans – Asynchronous result transmission

sub_type

The **sub_type** element represents a refinement of the recognition task. For address recognition purposes it might be necessary to supply the RC with the required coding depth like “inward or outward coding”. Basically the existence of the sub_type element and its value range depend on the specific capability.

examples for a address recognition request:

```
<simpleType name="CodingCapabilityT">
    <restriction base="NMTOKEN">
        <enumeration value="ROI"/>
        <enumeration value="outward"/>
        <enumeration value="inward"/>
        <enumeration value="street"/>
        <enumeration value="name"/>
    </restriction>
</simpleType>
```

symbology

The **symbology** depends on the characteristics of the info carrier. The data model provides fashioned data types for the different carriers. InfoCarrierT specifies the valid values.

```
<simpleType name="InfoCarrierT">
    <restriction base="NMTOKEN">
        <enumeration value="address"/>
        <enumeration value="barcode"/>
        <enumeration value="logo"/>
        ...
        <enumeration value="endln"/>
    </restriction>
    <!--
        address    info carrier is address
        barcode    a barcode
        logo       a logo
        endln      an endorsement line
    -->
</simpleType>
```

Note: The hyperlinks after the list entries refer to the description of the fashioned capability types. The following table provides the contiguity between info carrier, capability and symbology.

Info Carrier	Capability	Symbology
address	rec, sen	TextSymbologyT
barcode	dbc, cbc, pbc	BarcodeSymbologyT
logo	mcl	LogoSymbologyT
endln	end	TextSymbologyT

action	The action element influences the recognition of the RC. The MC can specify if the recognition is expected online, offline, or both on- and offline (mixed).
	
	`<simpleType name="ActionT">`
	`    <restriction base="NMTOKEN">`
	`        <enumeration value="online"/>`
	`        <enumeration value="offline"/>`
	`        <enumeration value="mixed"/>`
	`    </restriction>`
	`</simpleType>`

6.2.3.2.12 Region Type

This type is taken without change from CEN/TS 15448.

This type represents a template for different region descriptors specified in the base layer, e.g. address or barcode region descriptors.

```
<complexType name="RegionT">
    <sequence>
        <element name="type" type="mcrc:InfoCarrierT"/>
        <element name="symbology" type="string" minOccurs="0"/>
    </sequence>
</complexType>
```

Table 63

Element	Description

Type	provides information about the content of the region - e.g. *distribution code of recipient address*
	The element is linked with the simple type InfoCarrierT.
	`<simpleType name="InfoCarrierT">`
	`    <restriction base="NMTOKEN">`
	`        <enumeration value="address"/>`
	`        <enumeration value="barcode"/>`
	`        <enumeration value="logo"/>`
	`        <enumeration value="endln"/>`
	`    </restriction>`
	`    <!--`
	`        address    - Address Block Region`
	`        barcode    - Barcode Region`
	`        logo       - Region of a Logo`
	`        endln      - Endorsement Line Region`
	`    -->`
	`</simpleType>`

Symbology	The **symbology** is specified as string and tailored in the derived types dependent on the “type” element. For more details refer to the derived types specified in the base layer: address - refer to 6.2.3.4.1.1

NOTE RegionT and the types derived from RegionT are used in the submit mailpiece.

6.2.3.2.13 Batch Information Type

This type is used to transfer batch information at connection time.

```
<complexType name="BatchInformationT">
    <sequence>
        <element name="batch_info" type="mcrc:BatchDescriptionT"/>
    </sequence>
    <attributeGroup ref="mcrc:ProcessingConfigurationT"/>
</complexType>
```

Table 64

Element	Description
batch_info	For the definition of BatchDescriptionT see 6.2.3.4.2.

For the definition of the attribute group ProcessingConfigurationT see 6.2.3.1.7.

6.2.3.3 Customer Specific Domain Types

This subclause shall introduce in the creation of customer specific data types, which are usually derived from generic types. Tailored data types are either base types – refer to 6.2.3.4 – or project specific tailored types which are hosted in the customer types layer.

XML schema provides powerful object-oriented features. The schemas in this context use two deviation methods – deviation by restriction (which is primary used) and deviation by extension. Deviation by restriction comprises the deviation and redefinition of complex and simple types as well as the redefinition of elements. Deviation by extension corresponds to the extension of an inherited base type in an object-oriented programming language like Java or C++.

Why is the creation of derived and tailored types meaningful? It enables a user to tailor a generic type to its customer specific needs.

6.2.3.4 Domain Base Types

All domain base types are derived from generic types. The usual method is to derive by restriction. This means, that the derived type contains no new elements. Existing elements in the generic type are tailored – e.g. assignment of a new type – or omitted.

6.2.3.4.1 Region Types

The types in this subclause are derived from the generic type RegionT which consists of a ‘type’ element and the info carrier specific symbology.

RegionT is used when there is a need to provide the origin of a recognition result in a descriptive manner. The MC may for example request the rec_code capability from the RC. In the response the RC may provide recipient’s distribution code based on either the destination address or destination barcode. To distinguish between the two regions the element type in RegionT is used. The element symbology enables the result creator to provide the barcode type.

6.2.3.4.1.1 Region Type - Address

This type is taken without change from CEN/TS 15448.

AddressRegionT is derived from RegionT and specifies the valid content of an address region descriptor. This type is included in this schema only for the purposes of passing batch characteristics from the MC to the RC.

```
<complexType name="AddressRegionT">
    <complexContent>
        <restriction base="mcrc:RegionT">
            <sequence>
                <element name="type" type="mcrc:InfoCarrierT" fixed="address"/>
                <element name="symbology" type="mcrc:TextSymbologyT"/>
            </sequence>
        </restriction>
    </complexContent>
</complexType>
```

Table 65

Element	Description
Type	specifies content of region - fixed="address"

Element	Description

Symbology	TextSymbologyT provides the valid symbologies of an address label.
	
	`<simpleType name="TextSymbologyT">`
	`    <restriction base="NMTOKEN">`
	`        <enumeration value="machine"/>`
	`        <enumeration value="script"/>`
	`    </restriction>`
	`    <!--`
	`        machine    machine printed text - imprint`
	`        script     hand written text`
	`    -->`
	`</simpleType>`

6.2.3.4.2 Batch Description

BatchDescriptionT specifies the structure of a batch description. A batch is attached to a recipient address only - capability ***rec_code***.

```
<complexType name="BatchDescriptionT">
    <complexContent>
        <restriction base="mcrc:RecognitionResultT">
            <sequence>
                <element name="mailer_id" type="string" minOccurs="0"/>
                <element name="batch_id" type="string" minOccurs="0"/>
                <element name="code_attr" type="mcrc:BatchCodeAttrT" minOccurs="0"
                                                    maxOccurs="unbounded"/>
                <element name="status_attr" type="mcrc:BatchStatusAttrT" minOccurs="0"/>
                <element name="geo_mode" type=" mcrc:BatchCntryCodeAttrT " minOccurs="0"/>
                <element name="cntry_id" type="string" minOccurs="0"/>
                <element name="region" type="mcrc:AddressRegionT" minOccurs="0"/>
                <element name="location" type="mcrc:LocationT"/>
                <element name="restrictions" type="mcrc:RestrictionT" minOccurs="0"
                                                    maxOccurs="unbounded"/>
                <element name="mail_flow" type="string" minOccurs="0"/>
            </sequence>
        </restriction>
    </complexContent>
</complexType>
```

Table 66

Element	Description
Mailer_id	identification of the mailer of a particular batch
Batch_id	Identification of the individual batch
code_attr	**code_attr** provides information about features of the envelop, the address region and the address print itself.

```
<simpleType name="BatchCodeAttrT">
    <restriction base="NMTOKEN">
        <enumeration value="env_unk"/>
        <enumeration value="env_normal"/>
        <enumeration value="env_plastic"/>
        <enumeration value="pqua_std"/>
        <enumeration value="pqua_low"/>
        <enumeration value="pqua_colour"/>
        <enumeration value="bgr_std"/>
        <enumeration value="bgr_dark"/>
        <enumeration value="bgr_colour"/>
        <enumeration value="bgr_text"/>
        <enumeration value="bgr_unln"/>
        <enumeration value="fntt_var"/>
        <enumeration value="fntt_std"/>
        <enumeration value="fntt_italic"/>
        <enumeration value="fntt_dmatrix"/>
        <enumeration value="fntp_var"/>
        <enumeration value="fntp_prop"/>
        <enumeration value="fntp_const"/>
    </restriction>
    <!--
        - envelope characteristics
        env_var         - type of envelop varies
        env_normal      - type of envelop is normal
        env_plastic     - type of envelop is plastic
        - print quality
        pqua_std        - print quality of address is standard
        pqua_low        - low-contrast print quality of address
        pqua_colour     - coloured print of address
        - background characteristics
        bgr_text        - textured background in address block area
        bgr_unln        - background in address block area with underlines
        - info about the font type
        fntt_var        - font type varies
        fntt_std        - standard font type
        fntt_italic     - italic font
        fntt_dmatrix    - dot-matrix font
        - font pitch characteristics
        fntp_var        - font pitch varies
        fntp_prop       - proportinal font pitch
        fntp_const      - constant font pitch
    -->
</simpleType>
```

Element	Description
status_attr	**status_attr** provides information about the origin of the batch. BatchStatusAttrT specifies the valid ‘flags’.

	`<simpleType name="BatchStatusAttrT">`
	`    <restriction base="NMTOKEN">`
	`        <enumeration value="origin_unk"/>`
	`        <enumeration value="origin_sm"/>`
	`        <enumeration value="origin_mm"/>`
	`    </restriction>`
	`    <!--`
	`        origin_unk - origin of mail in the batch is unknown`
	`        origin_sm  - origin of mail in the batch is single mailer`
	`        origin_mm      - origin of mail in the batch is multiple mailer`
	`    -->`
	`</simpleType>`

geo_mode	This element provides information about the geographic destination of the mail batch and is of type "BatchCntryCodeAttrT" – refer to 6.2.3.4.3.
cntry_id	The ISO two-character country ID for the batch description, if the mail has a foreign destination ("export" for ***geo_mode***).

Region	The element provides information about the address region. AddressRegionT holds the info carriers' symbology which is essential for the batch processing.
	
	`<complexType name="AddressRegionT">`
	`    <complexContent>`
	`        <restriction base="mcrc:RegionT">`
	`            <sequence>`
	`                <element name="type" type="string" fixed="address"`
	`                                    minOccurs="0"/>`
	`                <element name="symbology"`
	`type="mcrc:TextSymbologyT"/>`
	`            </sequence>`
	`        </restriction>`
	`    </complexContent>`
	`</complexType>`
	
	refer to 6.2.3.4.1.1

location	location of a batch area – refer to 6.2.3.6
restrictions	Include or exclude zones – refer to 6.2.3.5.1
mail_flow	The **mail_flow** attribute specifies the mail flow currently processed. The values can be *outgoing* : mail is to be delivered all around the country (and outbound) *regional* : mail is to be delivered in the region of the sorting centre *incoming* : mail is to be delivered within the sorting centre area

6.2.3.4.3 Batch Destination

Provides information about the geographic destination of the mail batch.

```
<simpleType name="BatchCntryCodeAttrT">
    <annotation>
        <documentation>
  geographic mode chararcteristics: specifies the geographic sorting mode of the machine. The values can be:
    national - only national domestic mail is sorted
    import - mail coming only from foreign countries to the local country
    export - mail to be delivered only to foreign countries
    mix - all kinds of mail
        </documentation>
    </annotation>
```

```
        <restriction base="NMTOKEN">
            <enumeration value="national"/>
            <enumeration value="import"/>
            <enumeration value="export"/>
            <enumeration value="mix"/>
        </restriction>
    </simpleType>
```

6.2.3.5 Domain Primitives

The 'Domain Primitives' layer contains simple and complex data types, which are used for the specification of the more complex types in the higher level layers. Some of the primitives are independent of the domain like the string primitives. Other primitives have a strong relation to the postal domain.

NOTE Most of the simple types are self-explaining and not described in this section. Indefinite types or values should be documented in the XML schema listed in A.3.

6.2.3.5.1 Restriction Type

This type is taken without change from CEN/TS 15448.

The specification of restriction areas enables the MC to define include and exclude zones for the recognition.

```
<complexType name="RestrictionT">
    <sequence>
        <element name="type" type="mcrc:RestrictionTypeT"/>
        <element name="location" type="mcrc:LocationT"/>
    </sequence>
</complexType>
```

Table 67

Element	**Description**

Type	The **type** element specifies the kind of restriction. The valid scale of values is specified by simple type RestrictionTypeT.
	
	`<simpleType name="RestrictionTypeT">`
	`    <restriction base="NMTOKEN">`
	`        <enumeration value="include"/>`
	`        <enumeration value="exclude"/>`
	`    </restriction>`
	`</simpleType>`
	
	Include means that the RC has to examine the image inside the specified zone. If the zone information contradicts with the region of the requested capability (e.g. region hangs outside an inclusion zone), then the RC will return reject for the capability.
	
	Exclude zones are regions which are precluded from observation.
	
	Rules:
	
	1. There may be many inclusion zones and these may overlap.
	2. There may be many exclusion zones and these may overlap.

	3. Exclusion and inclusion zones may not overlap. 4. If at least one inclusion zone is defined, all locations must fall within an inclusion zone
location	The **location** element specifies the restriction zone.

6.2.3.5.2 Mailpiece Attributes Type

This type is taken from CEN/TS 15448 and extended by a “preknowledge” element.

The type *MailpieceAttrT* specifies basic mailpiece attributes.

```
<complexType name="MailpieceAttrT">
      <annotation>
            <documentation>Same type exists in OCR/VCS interface but withour preknowledge.
      Preknowledge will contain the base IC-ED RecognitionResultT structure              </documentation>
      </annotation>
      <sequence>
            <element name="type" type="mcrc:MailpieceTypeT"/>
            <element name="mailclass" type="string" minOccurs="0"/>
            <element name="faces" type="mcrc:FaceSetT" minOccurs="0"/>
            <element name="preknowledge" type="mcrc:RecognitionResultT" minOccurs="0"/>
      </sequence>
</complexType>
```

Table 68

Element	Description

type	defines the type of the mail item
	The simple type ***MailpieceT*** *provides the valid scale of values*
	
	`<simpleType name="MailpieceT">`
	`    <restriction base="NMTOKEN">`
	`        <enumeration value="unknown"/>`
	`        <enumeration value="letter"/>`
	`        <enumeration value="letter_bundle"/>`
	`        <enumeration value="flat"/>`
	`        <enumeration value="flat_bundle"/>`
	`        <enumeration value="parcel"/>`
	`    </restriction>`
	`</simpleType>`

mailclass	provides the mailclass of the mailpiece The valid range of mailclass values have to be customized project specific. This applies also to the code element in the mailclass result (MailClassT) which can be returned from an ED. Both – the mailclass element here and the code element as part of a mailclass result - have to provide the same range of values.

faces	The **faces** element is optional. The FaceSetT type is assigned to the element. Every **face** is linked with the corresponding **page attribute** which is required. **face** and **page** are important in an environment where multiple faces of a mail item have been lifted. If that is the case the IC has to provide the information about the disposable faces.
	
	`<complexType name="FaceSetT">`
	`    <sequence>`
	`        <element name="face" maxOccurs="unbounded">`

	`<complexType>`
	`  <simpleContent>`
	`    <extension base="mcrc:MailpieceFaceT">`
	`      <attribute name="page" type="unsignedInt"`
	`                 use="required"/>`
	`    </extension>`
	`  </simpleContent>`
	`</complexType>`
	`</element>`
	`</sequence>`
	`</complexType>`
	
	The valid scale of values of a face is specified in the type MailpieceFaceT, located in the primitives' layer.
	
	`<simpleType name="MailpieceFaceT">`
	`  <restriction base="NMTOKEN">`
	`    <enumeration value="top"/>`
	`    <enumeration value="bottom"/>`
	`    <enumeration value="left"/>`
	`    <enumeration value="right"/>`
	`    <enumeration value="front"/>`
	`    <enumeration value="back"/>`
	`  </restriction>`
	`</simpleType>`

preknowledge	Provides preknowledge to the RC System (see 6.2.3.2.1), e.g. a barcode result if a barcode reader is integrated into the machine. The structure for this result is RecognitionResultT

6.2.3.6 Region Location

This chapter is taken without changes from CEN/TS 15448. It is replicated here for the reader's convenience.

For object recognition the geometric description of a region is essential. A geometric region description represents an abstraction of an object located on an image.

To specify a region location, the region descriptor **LocationT** listed subsequently is used. **LocationT** encloses no information about the image orientation. The image orientation is specified in the TIFF header. Region coordinates are given in pixel and are related to the image origin which corresponds to the upper left corner. The coordinates are independent of the TIFF orientation.

```
<complexType name="LocationT">
    <sequence>
        <element name="face" type="mcrc:MailpieceFaceT" minOccurs="0"/>
        <element name="angle" type="mcrc:AngleRangeT" minOccurs="0"/>
        <element name="confidence" type="mcrc:ConfidenceRangeT" minOccurs="0"/>
        <element name="polygon" type="mcrc:PolygonT"/>
    </sequence>
    <attribute name="page" type="unsignedInt" use="optional"/>
</complexType>
```

The following table describes the attributes and elements of LocationT.

Table 69

Element	Description
attribute **page**	The **page** attribute is of vital importance, if an enrichment request provides a TIFF stream with multiple images. In that case the **page** attribute is mandatory. It enables the result provider to assign its result – the region - to a single image in the TIFF stream. The **page** attribute is optional, if the TIFF stream contains just one image. `<attribute name="page" type="unsignedInt" use="optional"/>` The region provider has to extract the **page** information from the TIFF header of the image the region relates to. The number is stored in the tag **PageNumber** (297).

face	**face** is basically optional. It is redundant in the result due to the fact, that the linkage between a region and the corresponding image is realized with the **page** attribute. **face** can represent a helpful hint in a multi-side scanning environment like a camera tunnel for a parcel applications. The valid **face** values are specified by MailpieceFaceT. The type is located in the primitives layer.
	
	`<element name="face" type="mcrc: MailpieceFaceT" minOccurs="0"/>`
	
	`<simpleType name="MailpieceFaceT">`
	    `<restriction base="NMTOKEN">`
	        `<enumeration value="top"/>`
	        `<enumeration value="bottom"/>`
	        `<enumeration value="left"/>`
	        `<enumeration value="right"/>`
	        `<enumeration value="front"/>`
	        `<enumeration value="back"/>`
	    `</restriction>`
	`</simpleType>`
	
	The assignment of the face information to the corresponding mail item is subject of 6.2.3.7

angle	The element **angle** specifies the orientation of the region. The element is of type float and accepts values between 0.0 and 360.0 degree. The clockwise rotation by this **angle** results in an oriented region (e.g. a horizontally oriented text line, which can be read from left to right or right to left in the case of mirrored images). The **angle** is measured from the x axis (== scan line) of the image to the read direction of a text line in the region.
	
	See Figure 34. Example for the handling of angle with mirrored regions.
	
	`<element name="angle" type="mcrc:AngleRangeT" minOccurs="0"/>`
	
	`<simpleType name="AngleRangeT">`
	    `<restriction base="float">`
	        `<minInclusive value="0.0"/>`
	        `<maxInclusive value="360.0"/>`
	    `</restriction>`
	`</simpleType>`

confidence	The element **confidence** provides a confidence level for the detected region. [float 0..1] – 1.0 highest confidence

polygon	The **polygon** of the region frame complies with a **right-angled rectangular** with 4 coordinate tuples (dots). The order of the corner marks are either counter-clockwise (in respect to the display of unmirrored images) or clockwise (in respect to the display of mirrored images) and indicate the orientation of the region. The dots correspond to the following corners:
	1st: top left
	2nd bottom left
	3rd bottom right
	4th top right
	No intersection is allowed.
	
	Note The upper left corner of a region relates to the regions content – e.g. the address text, text line or label - not to the frame. Figure B.10 shows an example which demonstrates this issue.
	
	`<element name="polygon" type="mcrc: PolygonT"/>`
	
	`<complexType name="PolygonT">`
	`    <sequence>`
	`        <element name="dot" minOccurs="4" maxOccurs="4">`
	`            <complexType>`
	`                <attribute name="x" type="int" use="required"/>`
	`                <attribute name="y" type="int" use="required"/>`
	`            </complexType>`
	`        </element>`
	`    </sequence>`
	`</complexType>`
	
	Note The polygon is not closed - the last coordinate is not the first one.
	
	The element name polygon is used to make changes to a more complex region description easier. If a future requirement enforces a more exact region description with more mesh points the occurrence qualifier of PolygonT still has to be changed. The element name stays unchanged.

Figure 33 and Figure 34 illustrate the description mentioned above.

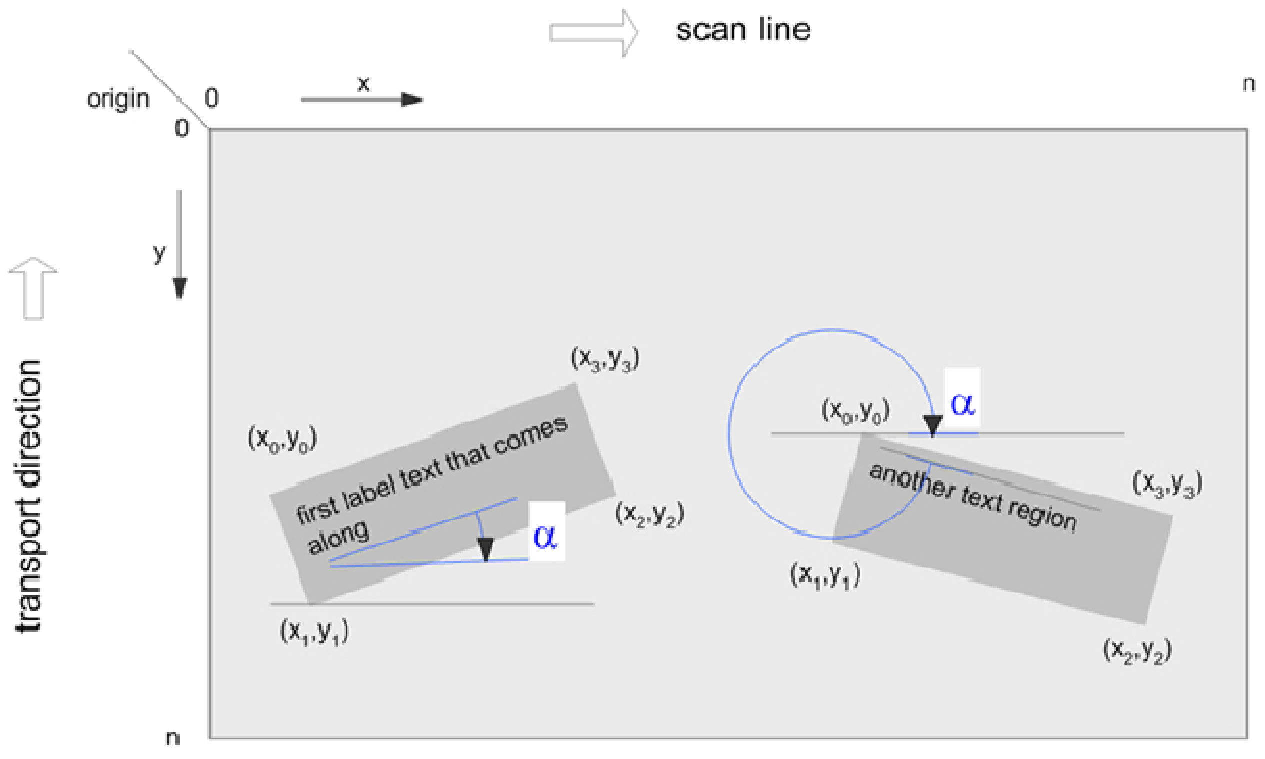


Figure 33 — Region Location Example 1

In the above example, it is shown how the anti-clockwise ordered co-ordinates describe the location and the orientation of the region. The angle (α) describes the rotation of the contents. When the region is rotated clockwise by this angle, the content will be horizontal. Note that the angle of the contents can differ from the angle of the region. Therefore, the angle described by the region co-ordinates may differ from the angle (α).

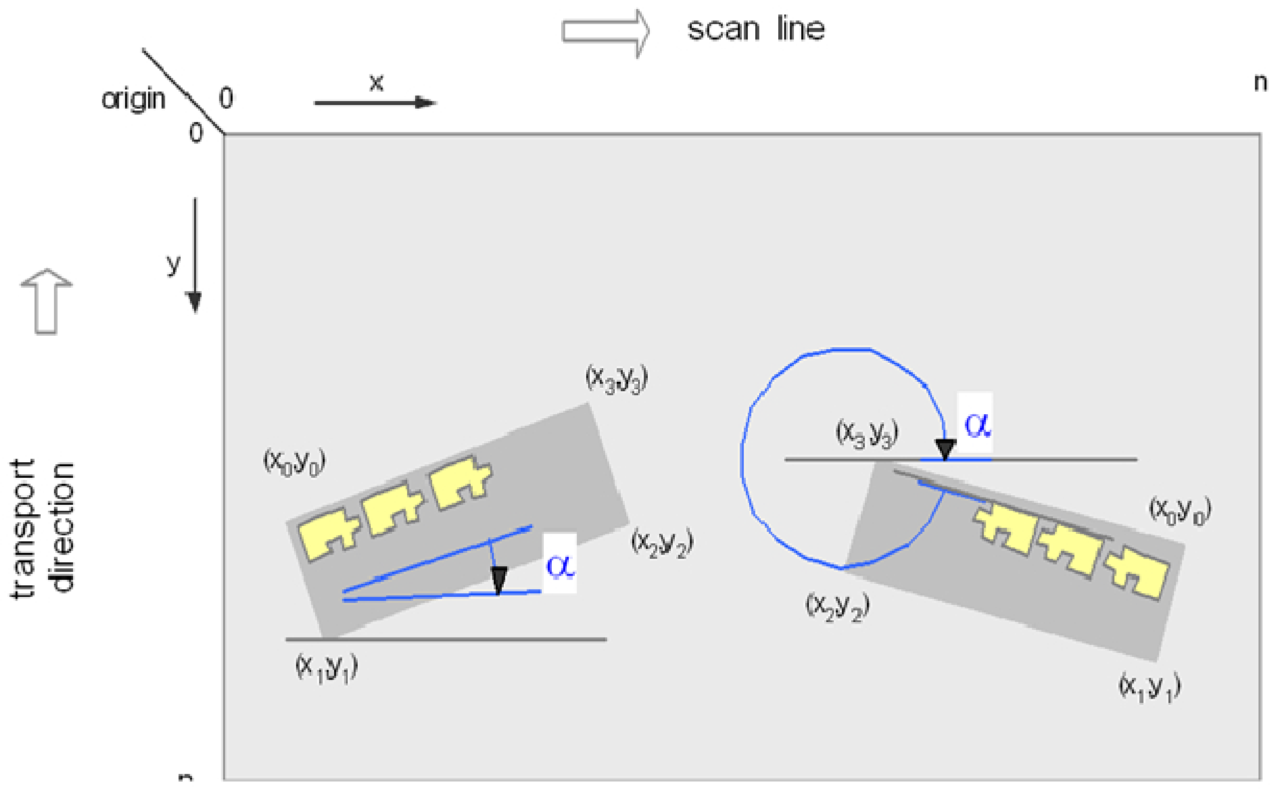


Figure 34 — Region Location Example 2

The above example shows how the co-ordinates are used to describe a mirrored region. The right hand region is mirrored (as shown by the shape). The co-ordinates are shown in a clockwise direction because the co-ordinates are given in the order top-left, bottom-left etc. When an attempt is made it put (x0y0) at the top left, then a “flip” must be performed to put x1y1 at the bottom left position. It is not possible to achieve this simply by rotating the region.

Note that the rotation (α) angle is related to the region prior to a “flip”. If the region is flipped, the angle (α) must be modified to account for the new rotation of the region. For example: If α = 20° in the region on the left side; then, assuming that the right side is a mirror image, α = 340° for the image on the right. If the region on the right is “flipped”, then the new angle is: α = 20° = (360° - 340°).

The region description is related to the image origin which is “upper left” in the shown examples. The TIFF header Orientation is **NOT** used when specifying coordinates.

The angle element is optional. Basically, it is redundant if the rotation angle of the surrounding rectangular matches the angle of the region content, e.g. the address block text. In some cases a clear match is not given – perhaps because the text lines in a region have different skew (e.g. a hand written address).

Figure 35 shows regions in different rotation angles to illustrate the sequence of the coordinates that specify the regions’ frame.

Figure 36 shows a text block. Every text line has its individual skew. The angle α represents the average or main skew of the region text.

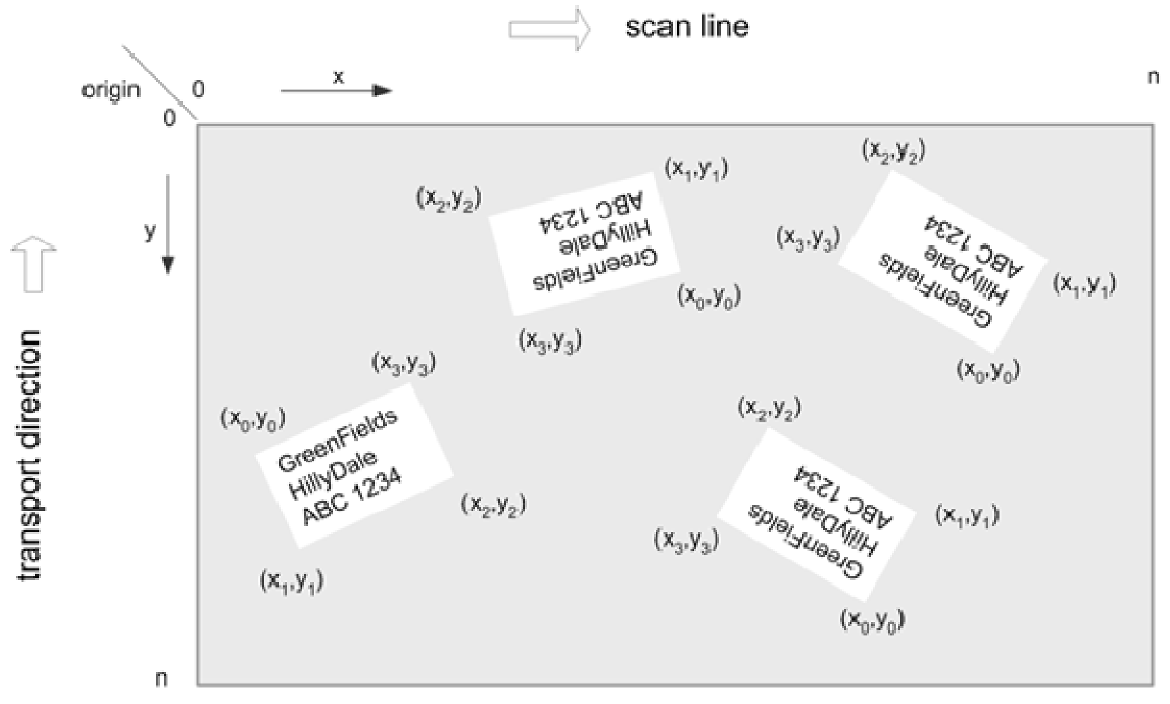


Figure 35 — Relation from Coordinates to Region Corner Marks

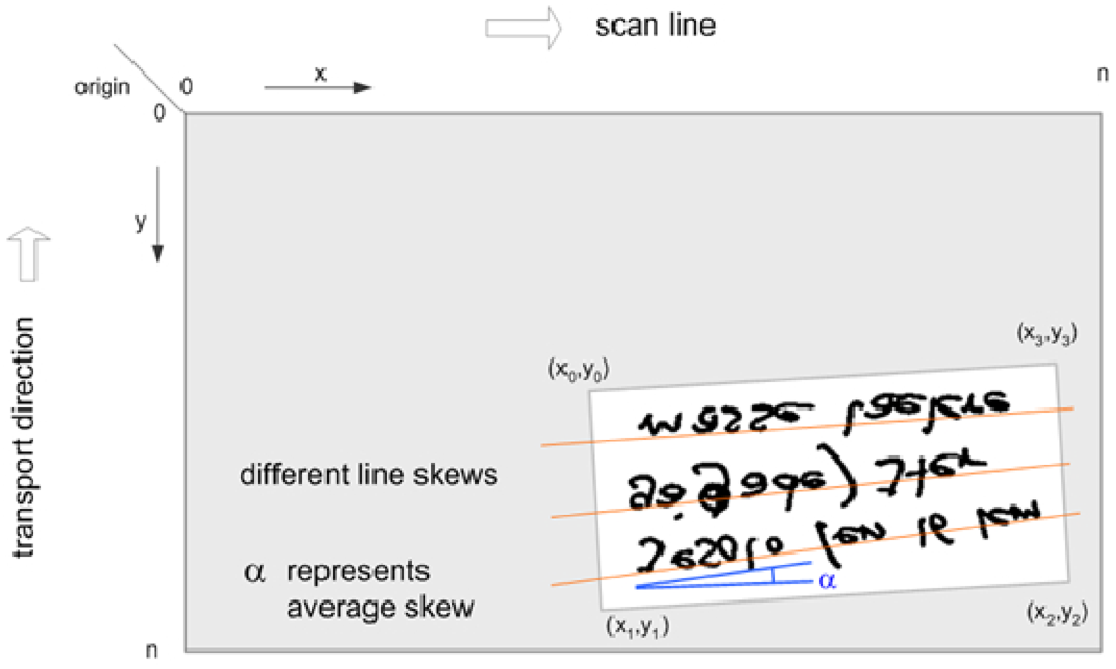


Figure 36 — Rotation Angle of a Script Block

6.2.3.6.1 Negative Coordinates and out-of-range Regions

To reduce complexity in the calculation of regions and their contents, the location is specified as a right-angled rectangle. This commitment leads to the phenomena of negative coordinates. This is an eye-catching thing but it is still a characteristic of the common case of out-of-range regions. Parts of the regions are located outside of the image, as illustrated in the following figure.

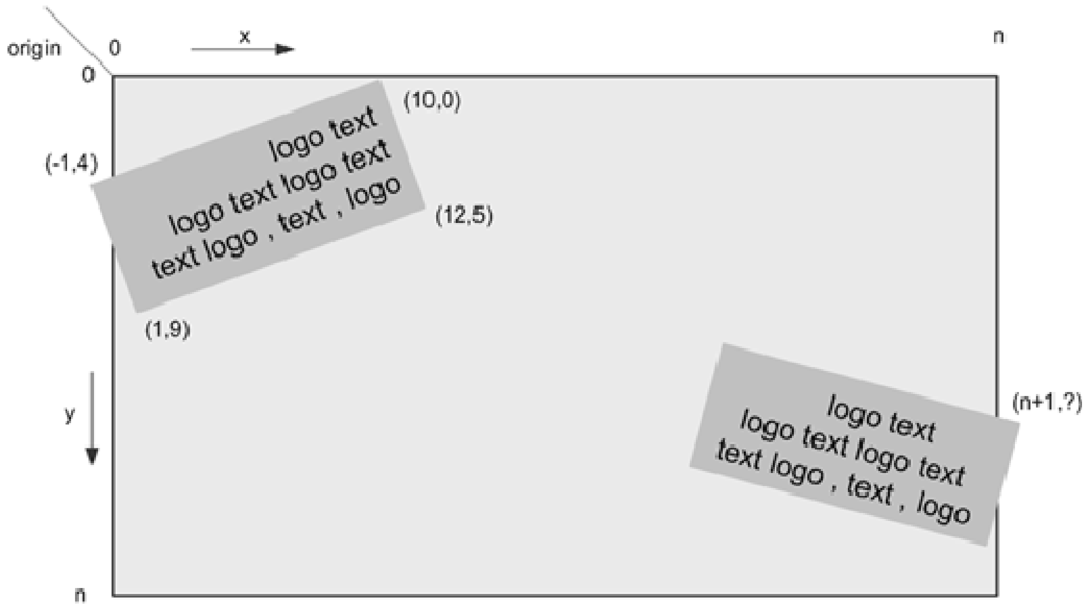
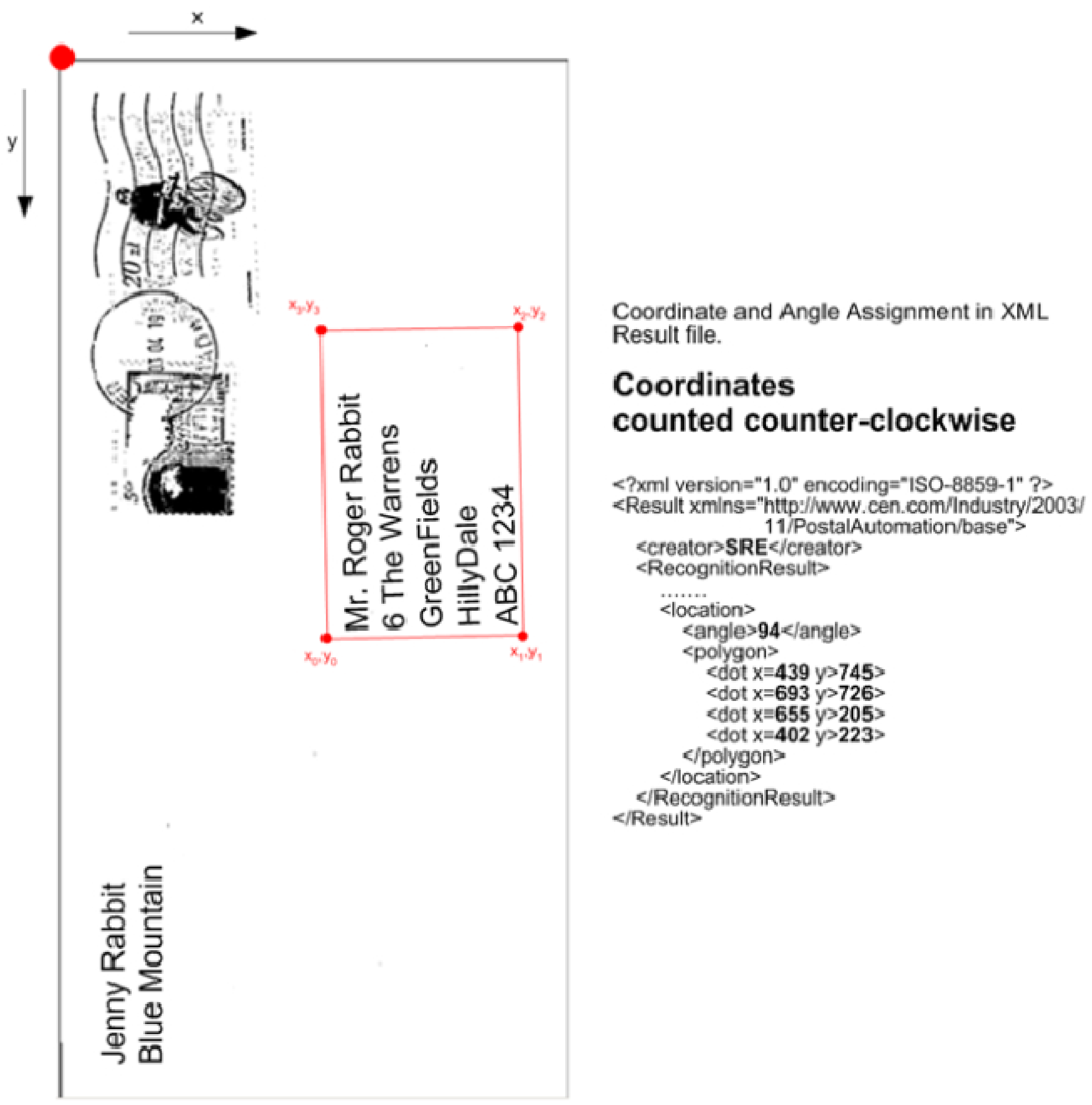


Figure 37 — Out-of-range Regions

6.2.3.6.2 Live-mail example

The following figure provides a live-mail example with the image and an extract of the related recognition result. face and page element are not specified because the TIFF stream in the request contained just a single image. The angle is specified with 94°. If the region is rotated clockwise by the angle, the address block is horizontally oriented and can be read from left to right.



Coordinate and Angle Assignment in XML Result file.

Coordinates counted counter-clockwise

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Result xmlns="http://www.cen.com/Industry/2003/
                  11/PostalAutomation/base">
   <creator>SRE</creator>
   <RecognitionResult>
      .......
      <location>
         <angle>94</angle>
         <polygon>
            <dot x=439 y>745>
            <dot x=693 y>726>
            <dot x=655 y>205>
            <dot x=402 y>223>
         </polygon>
      </location>
   </RecognitionResult>
</Result>
```

Figure 38 — Example with address block region

6.2.3.7 Assignment of Face Information

MailpieceFaceT specifies the six possible faces used to mark the sides of a mail item. This section specifies the assignment of face and side information.

The specification distinguishes between letter/flat and parcel application.

```
<simpleType name="MailpieceFaceT">
    <restriction base="NMTOKEN">
        <enumeration value="top"/>
        <enumeration value="bottom"/>
        <enumeration value="left"/>
        <enumeration value="right"/>
        <enumeration value="front"/>
```

```
            <enumeration value="rear"/>
        </restriction>
    </simpleType>
```

6.2.3.7.1 Letters and Flats

In the context of letter and flat applications exclusively the faces **front** and **back** are used. The assignment of the face information to the mail item side or scanning unit has to be specified in the project context.

front and **back** do not identify which is the front/back of the mailpiece (e.g. location of the address), but are used to identify the different physical sides of the mailpiece.

6.2.3.7.2 Parcels

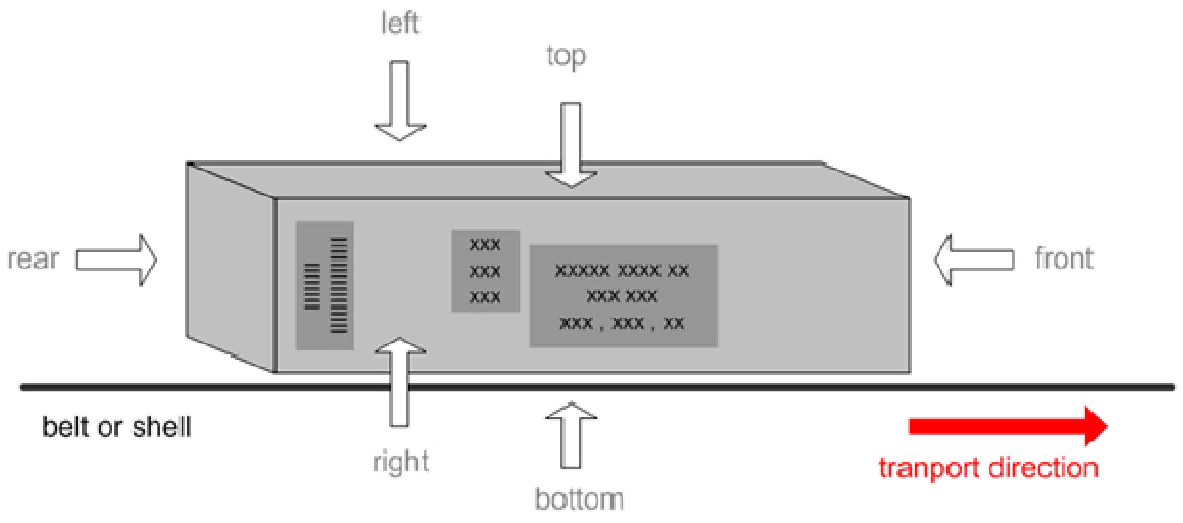


Figure 39 — Assignment of Face Information to Parcel sides

In Annex A a short introduction to XML and a sample XML Schema illustrates implementation.

7 System Design Description (SDD)

7.1 Overview

The System Design Description (SDD) clause defines in details the interface between the Machine Controller and Reading Coding System (MC/RC System Interface). It describes the architecture, the service model and behavioural model of the MC/RC System Standard interface.

The document is intended for all developers of the interface and it is a mandatory part of the Interface Standard along with the UCM and IDD sections, as depicted in Figure 40.

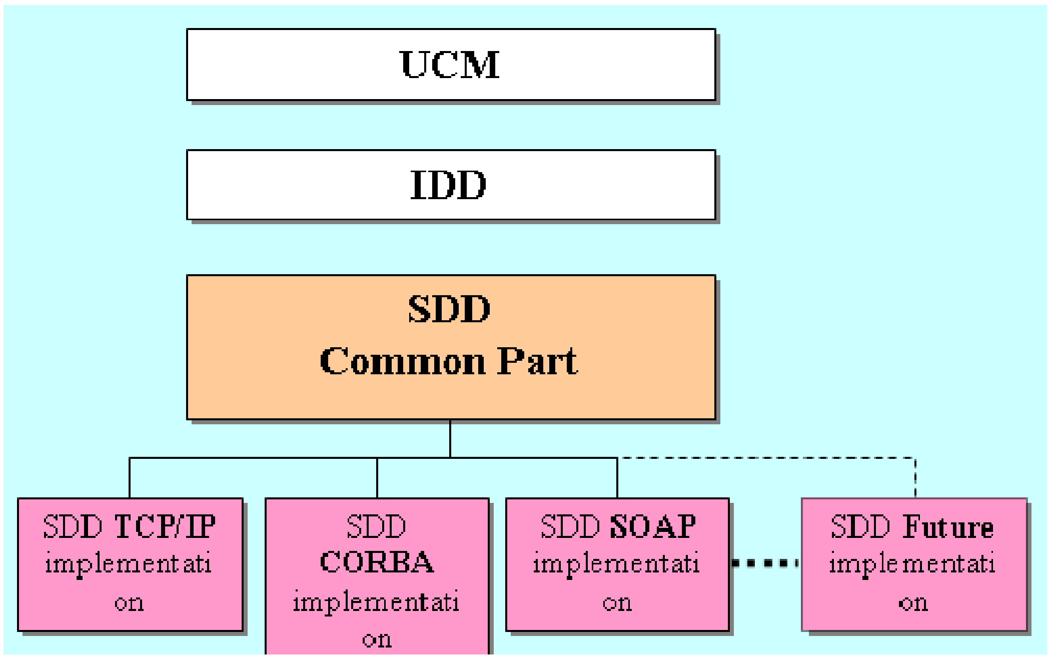


Figure 40 — MC-RC Interface Standard SDD Document Structure

In the Common Part of the SDD (this clause) no middleware or transport layer is specified. The common part of this document is intended to be middleware-independent.

In the specialized sections of Clause 8 (Middleware dependent parts of the SDD), the specifications are provided for three compatible transport solutions: TCP/IP, CORBA and SOAP. Further middleware solutions can be added when available, provided that they are fully compatible with the common part.

All the application data definitions, except for image data definition, are defined as part of the XML defined in the IDD in Clause 6. This includes the specific parameters of the Service calls.

7.2 Architectural Goals and Constraints

The system architecture considers mainly two components:

— The **Machine Controller (MC)** resident in the Machine, providing for mail flow control and interfacing with the external services.

— The **Reading Coding (RC)** System, providing for image data processing services and interfacing with the Machine(s).

The Image Processor (IP) can be resident in either the MC or the RC; these different configurations have no impact on the interface definition, but just at service call parameters level.

It is assumed that the system configuration is stable during the operations. The standard interface only deals with errors, exceptions and degrading of resources that may occur during live operations. Any system configuration change shall be performed and checked-out before starting the system operations. Configuration errors can still be detected in run-time but not automatically fixed or overcome through this interface specification.

7.2.1 Client-Server Model

A Client-Server model is used to model the relationship between the Machine Controller (MC) and the Reading Coding (RC) System.

The principle of the model is that the MC is the Client, which requests the Server (the RC System) to enrich mailpiece results. Hence, all exchanges have to be initiated by the MC (the Client), but there are the following exceptions:

— The RC System has to be able to inform the MC of its internal status change sporadically.

— The RC System returns the results asynchronously.

— The RC System may request the Machine status

— The RC System can perform a disconnection.

7.2.2 Client-Server Relationships

This standard defines the interface specifications for connecting one instance of the Machine Controller with one instance of the Reading Coding system (see Figure 41).

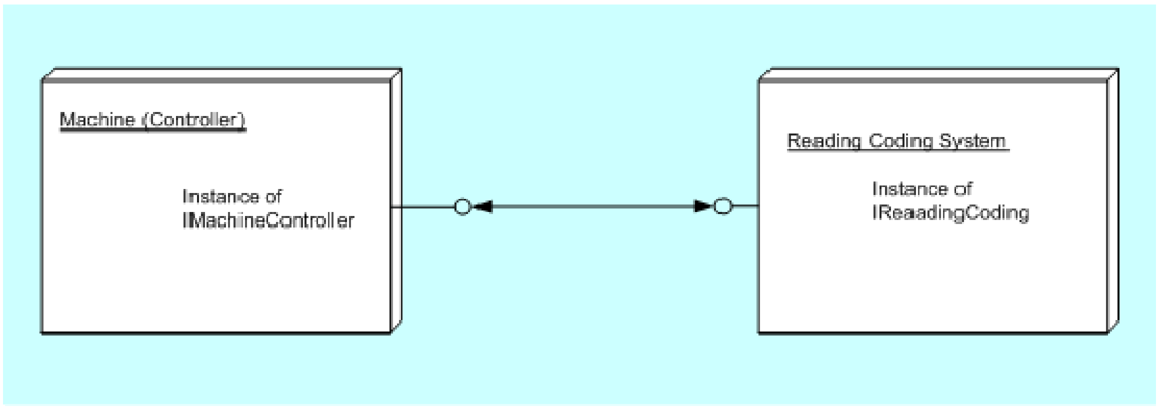


Figure 41 — Standard interface between one MC and one RC System

As it is specified in the Use Case Model (UCM) part the standard allows “m:n” relationship between Machines and RC systems using multiple instances of the interface. This means that:

a) A Machine can be connected to more than one RC System (see Figure 42).

b) More than one Machine can be connected to an RC System. (see Figure 43).

The diagrams below show these concepts graphically.

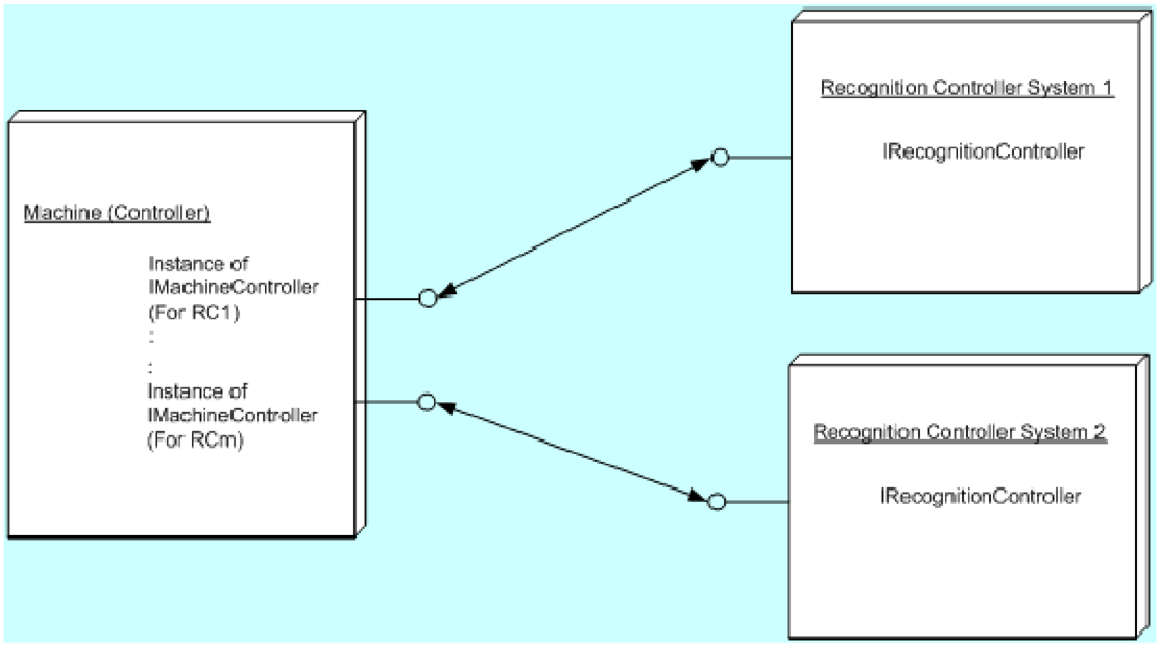

Recognition Controller System 1
IRecognitionController
Machine (Controller)
Instance of IMachineController (For RC1)
:
:
Instance of IMachineController (For RCm)
Recognition Controller System 2
IRecognitionController

Figure 42 — A Machine can be connected to several RC Systems (1 to m)

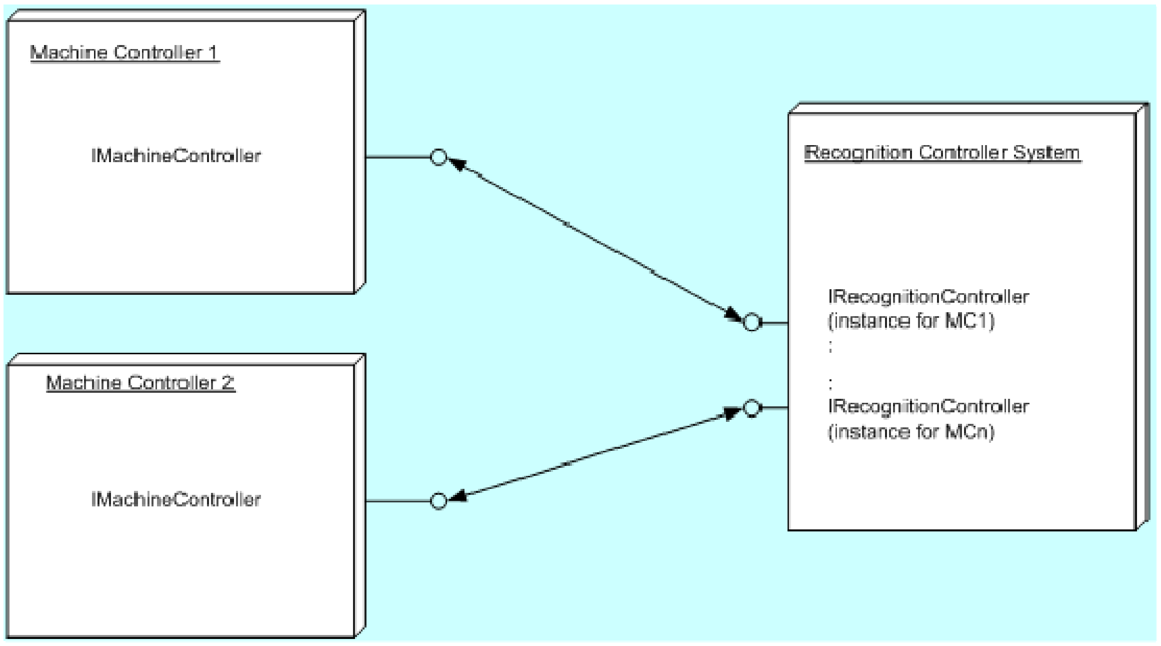

Machine Controller 1
IMachineController
Recognition Controller System
IRecognitionController (instance for MC1)
:
:
IRecognitionController (instance for MCn)
Machine Controller 2
IMachineController

Figure 43 — Several Machines (1 to n) can be connected to the same RC System

7.2.3 Server Selection

A Server Selection function shall be implemented for allowing the MC to select an RC server through a process that is middleware specific.

Then the MC can connect to the selected RC (through a virtual or physical address).

After the connection is established, the MC can negotiate the necessary RC capabilities and services.

The detailed description of this service is middleware-dependent and it is provided with each attached middleware-specific section.

A general schema is as follows:

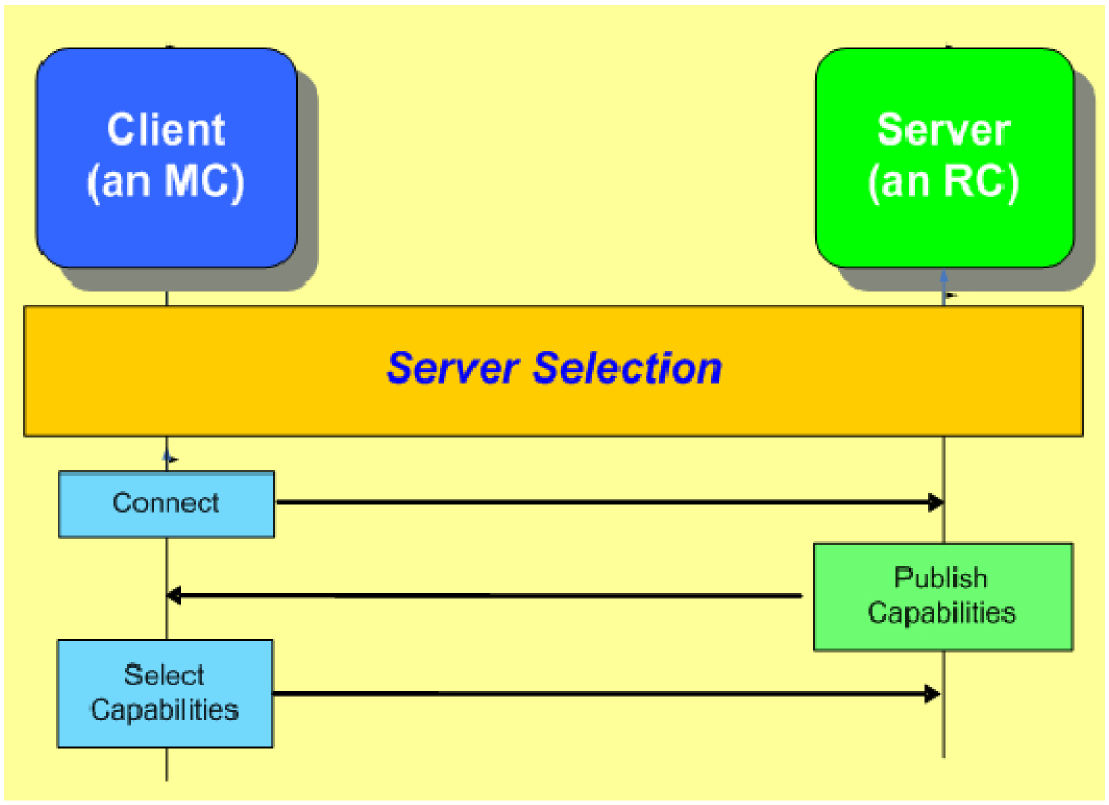


Figure 44 — General Schema of Server Selection

The timing is not defined here. As soon as the RC is physically connected to the communication network, the MC can access the network for a predefined sufficient time until the selected Server is found. Subsequently, the MC can connect to the required RC and negotiate capabilities. Timeouts and failures shall be considered, but their handling is application specific.

7.3 Service Model

7.3.1 Overview

This chapter defines in detail the interfaces that have been mentioned in the previous chapters. It defines the correct names of the interfaces, and supersedes all names used in the previous sections. The contents are also defined. Service diagrams are used.

The term “Service” identifies a system function in a middleware independent fashion; for example a web service or a CORBA method or a socket message.

Two interfaces are present and described in this standard to provide the sets of services used by the Machine and by the RC System:

— iMachineController

 is an interface which is implemented by the Machine Controller. The services implemented by the MC are used by the RC System.

— iReadingCoding

is an interface which is implemented by the RC System. The services implemented by the RC are used by the MC.

The following picture represents the two interfaces and the associated services that will be discussed in the following subclauses:

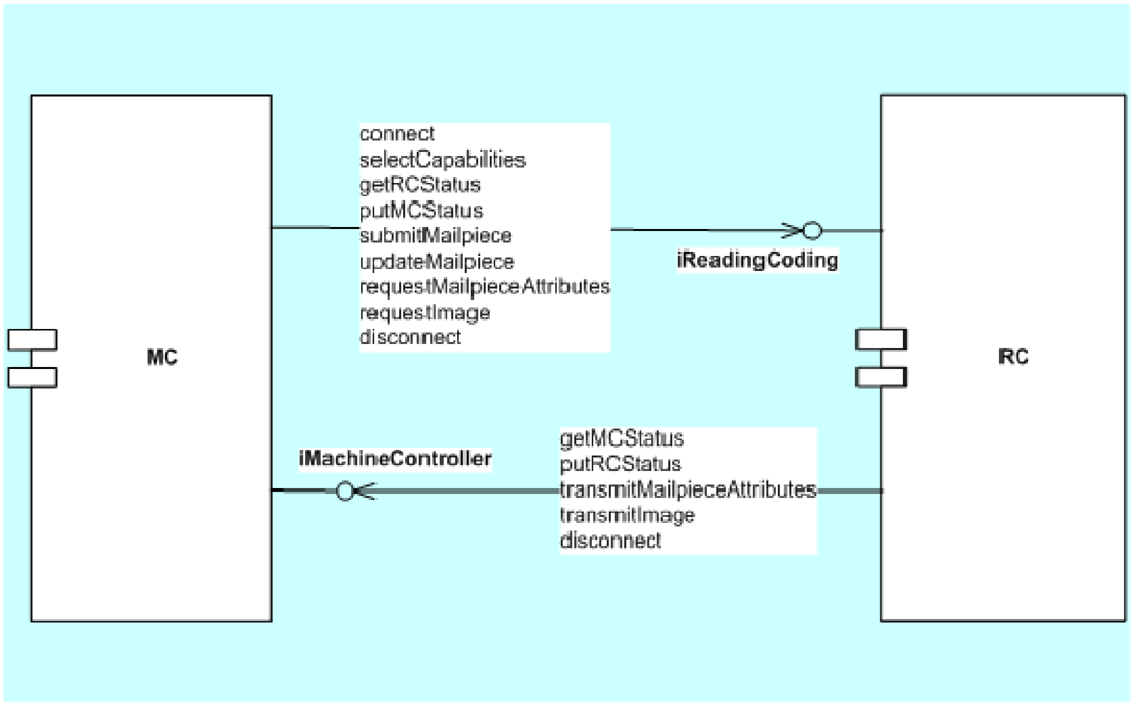


Figure 45 — Interfaces and Services

A Service call shall immediately return the control to the caller without return data. Each middleware implementation behaves differently; for example, CORBA returns the <void> value.

All Services are of the “no-waiting” type and the caller is responsible for handling a delayed response (regular condition, such as getXXStatus followed by putXXStatus within a predefined time period) or no response within a predefined time period (timeout condition).

Functions like putXXStatus can be issued asynchronously on event (i.e. the status change of the XX device).

In the following subclauses, each service call is described using a typical service description as follows:

<purpose brief description>
<pre-requisite(s)>
<input parameters list and their brief description> - application data
<resulting effect>
<exceptions>.

7.3.2 Exception Handling

Runtime errors often occur when a low-level piece of code is executing. The low-level functions shall not try to handle the exception themselves. Instead, they shall notify their caller of the error and let it decide how to handle it.

Exceptions are used to pass information about anomalous runtime conditions and to transfer control to an appropriate handler.

Depending on the middleware solution, there can be an exception thrown by a web service or CORBA method or a socket message.

Exceptions fall into the following main categories:

a) System level exceptions, which are middleware-specific (to be defined in the middleware-specific implementation).

b) Sender Application level exceptions, which are detectable by the initiating entity (MC or RC), without communicating to the receiving entity (to be defined with each specific application). Example: syntax error in a parameter for a message/method/service.

c) Receiving Application level exceptions which are detected at the receiving entity and communicated to the initiating entity via the return data of the reply message/method/service. This SDD clause will consider only these exceptions in its Common part.

The behaviour of the service calls initiated by the MC is as follows:

- connect results in putRCStatus with available RC capabilities and possible exceptions
- selectCapabilities results in no answer back
- submitMailpiece results in transmitMailpieceAttributes with possible exceptions
- updateMailpiece results in no answer back
- requestMailpieceAttributes results in transmitMailpieceAttributes with possible exceptions
- requestImage results in transmitImage with possible exceptions
- getRCStatus results in putRCStatus with possible exceptions
- putMCStatus results in no answer back
- disconnect results in no answer back

The behaviour of the service calls initiated by the RC is as follows:

- transmitMailpieceAttributes results in no answer back with possible exceptions
- transmitImage results in no answer back with possible exceptions
- putRCStatus results in no answer back
- getMCStatus results in putMCStatus with possible exceptions
- disconnect results in no answer back

7.3.3 Interface: IReadingCoding

This interface is implemented by the Reading Coding System in order to process the service calls issued by the MC. The implemented services enable for example, the MC to create a connection to the RC System and to submit mailpiece data for processing.

7.3.3.1 Service: connect

PURPOSE

This Service enables the MC (the client) to request a connection to the RC (the server).

Pre-requisite is the successful selection of the RC Server. Refer to the middleware-specific section for details on Server Selection.

The input parameters are:

— mcIfReference, reference to the MC standard interface

— XML data

The result is that the MC has successfully set-up a connection with the addressed RC. The confirmation is received via the putRCStatus service from the RC. The RC emits its status upon status change to “connected” with the list of all the available capabilities.

The MC can then proceed with selecting the required capabilities for image processing and have them granted by the RC.

EXCEPTIONS

— System Level (communication error)

— Application Level (invalid parameters or General Error)

The recovery actions are application specific. It is recommended to retry the connection request for a predefined number of times; in case of continuous failure, the MC should declare a severe error on the operating station of the Machine.

7.3.3.2 Service: selectCapabilities

PURPOSE

This Service is optional and it enables the MC to specify to the target RC the restricted sub-set of RC capabilities it intends to use while connected. If not issued, all the RC capabilities are intended to be used.

Pre-requisite is the successful execution of the connect service and the reception of the RC capabilities with the putRCStatus service.

The input parameters are:

— XML data containing the list of requested capabilities.

On return of the service call, the MC can use a granted (sub)set of RC capabilities.

With a putRCStatus the RC can notify to the MC problems such as:

— UnknownCapabilitySet – when an unpublished capability is requested,

- NoResources – when the RC has no available resources for one of the requested capability.

EXCEPTIONS

- System Level (communication error)
- Application Level (invalid parameters, RC not connected or General Error)

7.3.3.3 Service: getRCStatus

PURPOSE

This Service enables the MC to get the status of the RC to ensure that there is a connection between the Machine and the RC System. The MC shall issue this heartbeat request periodically when idle (no other service request issued within a predefined period). The heartbeat time period shall be specified in the Connect parameters.

Pre-requisite is a connection established between MC and one RC.

There are no input parameters.

The status of the RC System is returned asynchronously through the iMachineController::putRCStatus Service. In case of no answer within a pre-defined timeout period, the MC will assume that the RC is not active; the recovery of this situation is application specific; it is recommended to implement a retry strategy or re-connect.

EXCEPTIONS

- System Level (communication error)
- Application Level (invalid parameters, RC not connected or General Error)

7.3.3.4 Service: putMCStatus

PURPOSE

The MC informs the RC of its status if it has changed or in consequence of the getMCStatus request from the RC. This service call shall be immediately performed by the MC.

Pre-requisite is the active connection between the MC and the RC.

Input parameters:

- XML data carrying the MCStatusInfo (a structure which contains information regarding the current status of the Machine Controller).

EXCEPTIONS

- System Level (communication error)
- Application Level (invalid parameters, RC not connected or General Error)

7.3.3.5 Service: submitMailpiece

PURPOSE

This Service enables the MC to transfer a coding service request to the Reader Coding System. This service call enables the MC to request the RC process the data according to all or some of the available capabilities. The Service provides for including TIFF image data as an input parameter. This Service can be executed multiple times by the MC as long as the mailpiece is under processing in the Machine.

Pre-requisites are: MC connected to the RC; RC's capabilities set achieved by the MC.

Input Parameters (the main data transported; see detailed and complete definition in the IDD document and in the middleware implementation specific section):

- XML data containing: the mailpiece ID, the time in milliseconds within which the RC has to provide a result, the mailpiece attributes (for example: recognition results already made available by the Machine capabilities); the requested RC capabilities;
- the TIFF image (optional). It is defined as a sequence of images (one or many); it can also be an empty parameter just to inform the RC system that a new item is under processing (the image data are processed inside the Machine);

The RC will process the mailpiece data using the requested set of capabilities and it will return the results to MC when done (or whatever is available when the timeout expires) using the IMachineController::transmitMailpieceAttributes service.

The RC can report to the MC error conditions such as:

- InvalidID – the mailpiece identification is out of range or duplicated
- NotSupportedCapability – the RC does not support the requested capabilities (this should not happen since the capabilities have been previously negotiated)
- Saturated – The mailpiece cannot receive service from the RC (the device is saturated or busy) and it will be discarded; a recovery strategy is application-specific and it will not be defined in this standard;
- NoResources – the RC cannot process the mailpiece data due to lack of specific resources;

EXCEPTIONS

- System Level (communication error)
- Application Level (invalid parameters, RC not connected or General Error)

7.3.3.6 Service: updateMailpiece

PURPOSE

This Service enables the MC to transfer updated or additional information related to a mailpiece that has already been submitted to the RC System. This Service can be executed multiple times by the MC as long as the mailpiece is under processing in the Machine.

For a sorting pass without image lift (e.g., inward sort or sequence sort) this Service can be issued to provide the bin number to which a mail piece is physically sorted; for this case there may be no prior submitMailpiece.

Pre-requisites are: MC connected to the RC.

Input parameters (main data transported; see detailed and complete definition in the IDD document):

— XML data containing the updated mailpiece attributes, including the mailpiece identification.

The RC will update the database entry for that mailpiece with the information transmitted by the MC. No answer back data from RC are expected with this service call.

EXCEPTIONS

— System Level (communication error)

— Application Level (invalid parameters, RC not connected or General Error)

7.3.3.7 Service: requestMailpieceAttributes

PURPOSE

This Service enables the MC to request mailpiece attributes from the RC System. This is the typical case of coding data retrieval based on an ID-tag read on the mailpiece.

Pre-requisites are: MC connected to the RC.

Input parameters (main data transported; see detailed and complete definition in the IDD document):

— XML data indicating the unique mailpiece identification (ID-tag).

The result will be returned by the RC through the IMachineController::transmitMailpieceAttributes service call. All the available attributes will be returned.

The RC can report to the MC error conditions such as:

— InvalidID – the mailpiece specified by the mailpiece ID is not currently known by the RC.

EXCEPTIONS

— System Level (communication error)

— Application Level (invalid parameters, RC not connected or General Error)

7.3.3.8 Service: requestImage

PURPOSE

This Service enables the MC to request an image (set) from the RC. This service is used to retrieve the image (set) of the most recently lifted image at a reading station. Typical applications supported are: the Machine displays the image for address block cropping or for diagnostic checking.

Pre-requisites are: MC connected to the RC.

Input parameters (main data transported; see detailed and complete definition in the IDD document):

- XML data indicating a specific Feeder ID and Scanner ID, image selection by type and resolution.

The image set that meets all the selection criteria will be returned to the requesting MC by the RC through the IMachineController::transmitImage service in TIFF format.

The RC can report to the MC error conditions such as:

- InvalidID – the RC does not recognize the feeder ID and/or the scanner ID.
- NoImages – the RC has no images that meet the specified selection criteria (Feeder ID, Scanner ID, type, resolution).

EXCEPTIONS

- System Level (communication error)
- Application Level (invalid parameters, RC not connected or General Error)

7.3.3.9 Service: disconnect

PURPOSE

This Service enables the MC to force disconnection of the target RC System.

Pre-requisite is an existing connection between the MC and the RC.

There are no input parameters.

No answer back from the RC is expected.

EXCEPTIONS

- System Level (communication error)
- Application Level (invalid parameters, RC not connected or General Error)

7.3.4 Interface: IMachineController

This interface is implemented by the Machine Controller in order to process the service calls issued by the RC. The Services are called by the RC and enable for example, the RC to return mailpiece data processing results or to inform the MC for changes of its internal state.

7.3.4.1 Service: getMCStatus

PURPOSE

This Service enables the RC to get the status of the MC to ensure that there is a connection between the Machine and the RC System.

There are no input parameters.

The state of the RC System is returned in response through the IReadingCoding::putMCStatus service call.

EXCEPTIONS

- System Level (communication error)
- Application Level (invalid parameters, RC not connected or General Error)

7.3.4.2 Service: putRCStatus

PURPOSE

In response to getRCStatus and connect issued by the MC or on RC status change, this Service allows the RC to report its status to the MC, mainly that it is connected and either ready or not to provide services according to its capabilities. It is also used to indicate that the RC server is terminating and is no longer available to the MC.

The input parameters are:

- XML data including the “RCStatusInfo” structure, defining the current status of the RC System and the status of the selected capabilities (all capabilities unless restricted via the selectCapabilities service).

EXCEPTIONS

- System Level (communication error)
- Application Level (invalid parameters, RC not connected or General Error)

7.3.4.3 Service: transmitImage

PURPOSE

This Service enables the RC to transfer an image set to the Machine Controller in response to a IMachineController::requestImage Service call. The selection criteria specified in the originating service call (Feeder ID, Scanner ID, type, resolution) are applied. This function is provided in order to support the diagnostic display of images and the address block coordinates selection at the Machine console.

The input parameters are:

- XML containing the full set of associated mailpiece attributes, an indicator of the image status, any reported error condition.
- the TIFF container, defined as a sequence of images (one or many).

The RC retrieves the requested image (set) from an internal storage and transfers it to the requesting MC.

The RC can report to the MC error conditions such as:

- InvalidID – the RC does not recognize the feeder ID and/or the scanner ID.
- NoImages – the RC has no images that meet the specified selection criteria (Feeder ID, Scanner ID, type, resolution).

EXCEPTIONS

- System Level (communication error)
- Application Level (invalid parameters, RC not connected or General Error)

7.3.4.4 Service: transmitMailPieceAttributes

PURPOSE

In response to submitMailpiece or requestMailpieceAttributes issued by the MC, this Service enables the RC to transfer (all available) mailpiece attributes to the Machine Controller.

The input parameters are:

— XML data, including all the available attributes for the mailpiece, the status of the result and any possible error condition.

The RC can report to the MC error conditions such as:

— InvalidID – the mailpiece specified by the mailpiece ID is not currently known by the RC.

EXCEPTIONS

— System Level (communication error)

— Application Level (invalid parameters, RC not connected or General Error)

7.3.4.5 Service: disconnect

PURPOSE

This Service enables the RC to force disconnection of the MC.

There are no input parameters.

No answer back from the RC is expected.

EXCEPTIONS

- System Level (communication error)
- Application Level (invalid parameters, RC not connected or General Error)

7.3.5 Data definition

All data definitions can be found in the IDD (clause 6) and in the middleware specific section.

7.4 Behavioural Model

The ideal behaviour of each function, in a scenario where no exception handling is required, shall be compliant with the base flow diagrams described in the Uses Cases clause above.

The exception handling is an alternative flow that is not described in this document since it is to be specified in the application environment.

A general state sequence is as follows:

1) MC to find the RC Server and its address (server selection phase, middleware-dependent).
2) MC to Connect to the RC; the MC retries until successful or it declares failure/stop; alternatively, the MC can connect to another RC; the MC receives the full list of the RC capabilities that are considered as selected by default.
3) MC to optionally select the subset of required RC capabilities; on exception, the MC retries or it disconnects and goes back to connect (1).
4) Process Mailpieces; on exception, the caller device retries or disconnects and goes back to connect (1).
5) Exchange Status; status information is emitted asynchronously when the status changes.
6) Disconnect and get ready to connect again (1) or stop.

The Basic Flow Diagrams are provided for the following functions:

- Server Selection – middleware-dependent – see middleware-specific sections (Clause 8)
- Connect and Capabilities – common – see 5.3.2
- Image Processing – common – see 5.3.3
- Status - common – see 5.3.10 and 5.3.12
- Disconnect - common – see 5.3.3

8 Middleware dependent parts of the SDD

8.1 SDD TCP/IP Implementation

8.1.1 Communication layer

The interface is based on Ethernet standard component as defined in ISO/IEC 8802-3, ANSI/IEEE 802.3, IEEE 802.1x and IEEE 802.2.

The software protocols used are IP & TCP as defined in RFCs 791, 793 and 894. Other related protocols may be used by the network layer of the computers, like ICMP (RFC 792), ARP (RFC 826).

Communication between the Machine and the RC System is based on standard TCP/IP sockets over Ethernet.

A single bidirectional port is used for communication.

8.1.2 Server selection

IP address plan and port numbers are defined by the project. The server selection is done by the IP address and the port number.

Purpose

This procedure enables the client (Machine) to request a server (RC System) for a new connection.

On the RC System side, the system shall publish its presence. It consists in accepting TCP connection in server mode.

On the Machine side, the system looks for the IP address and the TCP port of the server.

The parameters of the connection are:

- the IP address of the RC System
- the TCP port number

8.1.3 Message definition

All messages are composed of a message length followed by the message body. The message length gives the number of bytes of the message body (i.e. 'message length' does not include itself).

The message structure is described by a four-column table:

— The first column gives the field position measured in bytes with respect to the message body;

— The second column gives the type of field (see below);

— The third column gives the size of field (in bytes);

— The last column gives the meaning of the field and the possible values, if applicable.

The data can be of the following types:

- INT32 for a 32-bit word, in standard network-order (i.e. big-endian convention, with the Most Significant Byte sent first, also known as 'network-order');
- CHAR[] for a null-terminated character string;
- BYTE[] for an array of bytes, whose length is dynamically given by a preceding field;
- BYTE[4] for an array of 4 bytes;
- BYTE[0..3] for 0 to 3 padding bytes needed to align the next structure.

Table 70

Offset	Type	Size	Description
---	INT32	4	`MSG_LENGTH`
0	INT32	4	`XML_SIZE`
4	BYTE[4]	`4`	`XML_TAG`: tag identifying that the next field is the XML string buffer. This tag is equal to `0x58 0x4D 0x4C 0x20` (i.e. '`XML `' with an added space character for padding).
8	CHAR[]	`XML_SIZE - 4`	`XML_BUFFER`
XML_SIZE + 4	BYTE[0..3]	0 to 3	`XML_PADDING`: 0 to 3 null bytes length, null if `XML_SIZE` is multiple of 4 `4 – (XML_SIZE mod 4)` otherwise.
offsetData0	INT32	4	`DATA_0_SIZE`. If 0, this is the last field of the message.
If DATA_0_SIZE is zero, the following fields do not exist.			
offsetData0 + 4	BYTE[]	`DATA_0_SIZE`	`DATA_0_BUFFER` (if `DATA_0_SIZE!=0`).
offsetData0 + 4 + `DATA_0_SIZE`	….	0 to 3	`DATA_0_PADDING` 0 to 3 null bytes length, null if `DATA_0_SIZE` if multiple of 4 `4-(DATA_0_SIZE mod 4)` otherwise.
offsetData_1	INT32	4	`DATA_1_SIZE`
If DATA_1_SIZE is zero, the following fields do not exist.			
offsetData_1 + 4	BYTE[]	`DATA_i_SIZE`	`DATA_1_BUFFER`
	BYTE[0..3]	0 to 3	`DATA_1_PADDING`
	INT32	4	`DATA_2_SIZE` = 0

The DATA_i_BUFFER fields are used for image buffers in TIFF format.

The XML_BUFFER data contains the message written in XML in coherence with the XSD schema defined in the IDD.

Table 71

Message list	Acknowledgment	Exceptions
connect	… putRCState due to state change	No response
selectCapabilities	… putRCState due to state change	No reponse
getRCState	putRCState	putRCState
putMCState	No response	No response
submitMailpiece	transmitMailPieceAttributes (deferred and possibly on another connection…)	transmitMailPieceAttributes
updateMailpiece	No Response	No Response
requestMailpieceAttributes	transmitMailPieceAttributes	transmitMailPieceAttributes
requestImage	transmitImage	transmitImage
disconnect	No response	No response

Table 72

Message list	Acknowledgment	Exceptions
getMCState	putMCState	putMCState
putRCState	No response	No response
transmitImage	No response	No response
transmitMailPieceAttributes	No response	No response
disconnect	No response	No response

The common part of the SDD defines System level exceptions which are middleware-specific. For TCP/IP implementation, they could be socket read errors, socket write errors, socket shutdown. In case of System level exception, the system shall operate a TCP disconnection.

8.2 SDD CORBA Implementation

This chapter describes the CORBA implementation of the SDD.

8.2.1 Server Selection

The CORBA Naming Service will be used for the server selection. RC Systems publish their presence in the naming service and the MC discovers an RC System by a lookup. Figure 46 shows the general structure of the naming service context and an example of naming service entries. The green nodes contain the actual CORBA references to IReadingCoding

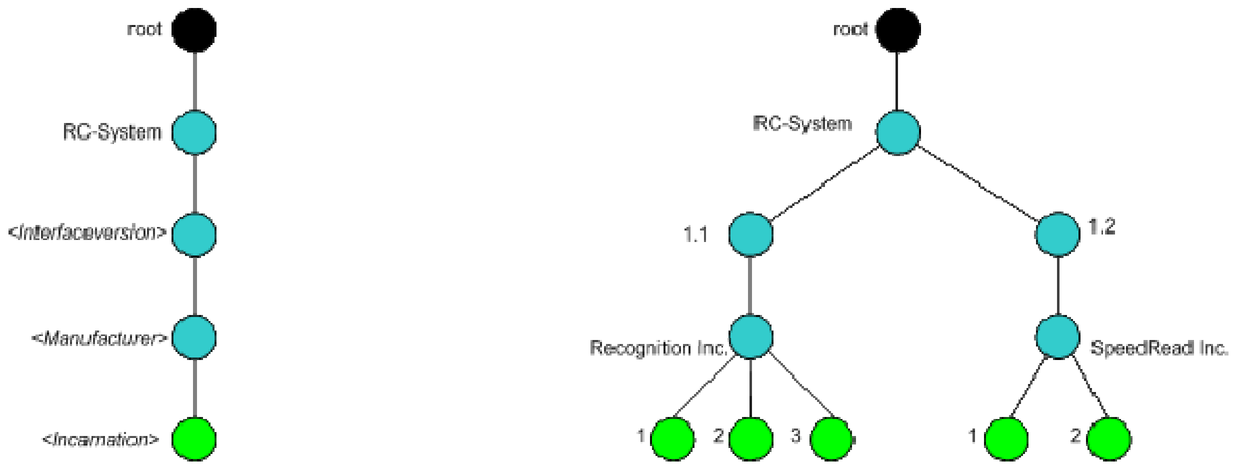


Figure 46 — General Structure (left) and Example (right) of CORBA Naming Service

The relevant naming service context has the following levels:

— the common level “RC-System”

— the interface version (format “X.Y”, X is major number, Y is minor number)

— the manufacturer of the RC System

— the incarnation of the interface reference

The RC System is responsible for creating the relevant naming service context if it is not yet available.

The permanent communication errors in the middleware throw the CORBA exception COMM_FAILURE.

8.2.2 ORB Implementation

There are a variety of ORB implementations on the market. In general, interoperability cannot be guaranteed. However, experiences from the vendors involved in the definition of this interface show, that the free ORB (Object Request Broker) implementations OmniORB, TAO and JacORB interact without problems. At minimum ORBs implementing CORBA version 2.4 shall be used. The use of ORBs implementing higher CORBA versions is also possible, but of course, only downward compatible parts of the implementation must be used.

8.2.3 Interface Definition

In the following, we use IDL for the concrete definition of the interfaces. Each service from the common part is mapped to a CORBA method of the same name in the IDL. The data sent with the single CORBA methods, which is common to all Implementations, is defined in the common part. In the IDL it is referred to as xmlData.

8.2.3.1 Interface: IReadingCoding

The following IDL defines the CORBA implementation of the interface IReadingCoding.

```
                         <<interface>>
                         IReadingCoding

connect       (
               in string: xmlData
               in IMachineController: mcIfReference) : void
         raises (ConnectionRejected);

selectCapabilities(
               in string: xmlData) : void
         raises (NotConnected, GeneralError);

getRCStatus   () : void
         raises (NotConnected, GeneralError);

putMCStatus   (
               in string: xmlData) : void
         raises (NotConnected, GeneralError);

submitMailpiece   (
               in string: xmlData
               in TiffMpImageList: image) : void
         raises (NotConnected, GeneralError);

updateMailpiece   (
               in string: xmlData) : void
         raises (NotConnected, GeneralError);

requestMailpieceAttributes(
               in string: xmlData) : void
         raises (NotConnected, GeneralError);

requestImage  (
               in string: xmlData) : void
         raises (NotConnected, GeneralError);

disconnect    () : void
         raises (NotConnected, GeneralError);

                           Exceptions

exception ConnectionRejected
{
    string errDescription;
};
exception NotConnected
{
    string errDescription;
};
exception GeneralError
```

```
{
    string errDescription;
};
```

8.2.3.2 Interface: IMachineController

The following IDL defines the CORBA implementation of the interface IMachineController.

```
                         <<interface>>
                      IMachineController

getMCStatus   () : void
         raises (NotConnected, GeneralError);

putRCStatus   (
               in string: xmlData) : void
         raises (NotConnected, GeneralError);

transmitImage (
               in string: xmlData
               in TiffMpImageList: image) : void
         raises (NotConnected, GeneralError);

transmitMailpieceAttributes (
               in string: xmlData) : void
         raises (NotConnected,GeneralError);

disconnect    () : void
         raises (NotConnected, GeneralError);

                          Exceptions

exception NotConnected
{
    string errDescription;
};
exception GeneralError
{
    string errDescription;
};
```

8.2.3.3 Data Types

This subclause defines the data types and structures, which are mentioned in the above subclauses.

```
// Image data in TIFF format
// Description:
// The TIFF image binary data.
```

```
typedef sequence<octet> TiffMailpieceImage;
typedef sequence<TiffMailpieceImage> TiffMpImageList;
```

8.3 SDD SOAP Implementation

This subclause describes the SOAP implementation of the SDD for the MC/RC Interface Standard Specification.

Simple Open Access Protocol (SOAP) is a simple XML based protocol to let applications exchange information based on XML and HTML. SOAP is a specification that details how to encode information and messages used in Web Services.

The MC/RC Interface is described using WSDL (Web Services Description Language) and its XML expansion as generated by XML Spy.

All the WSDL used in this document are compliant with following standard specifications:

— http://www.w3.org/2001/XMLSchema (XML Schema namespace)

— http://schemas.xmlsoap.org/wsdl/soap (SOAP namespace)

8.3.1 Server Selection

8.3.1.1 Basic Flow Diagram

It is assumed that the Machine Controller and the Reading Coding system are devices connected to the same communication network.

A link between the MC and the RC system, using pre-defined static configuration information, must be established before any information may be exchanged.

The RC validates the server selection request from the MC. Validation covers the checking of the interface version, number of connected MCs (for licensing), etc.

The RC can reject a selection request (licensing issues).

The complete server selection phase is described in the sequence diagram below.

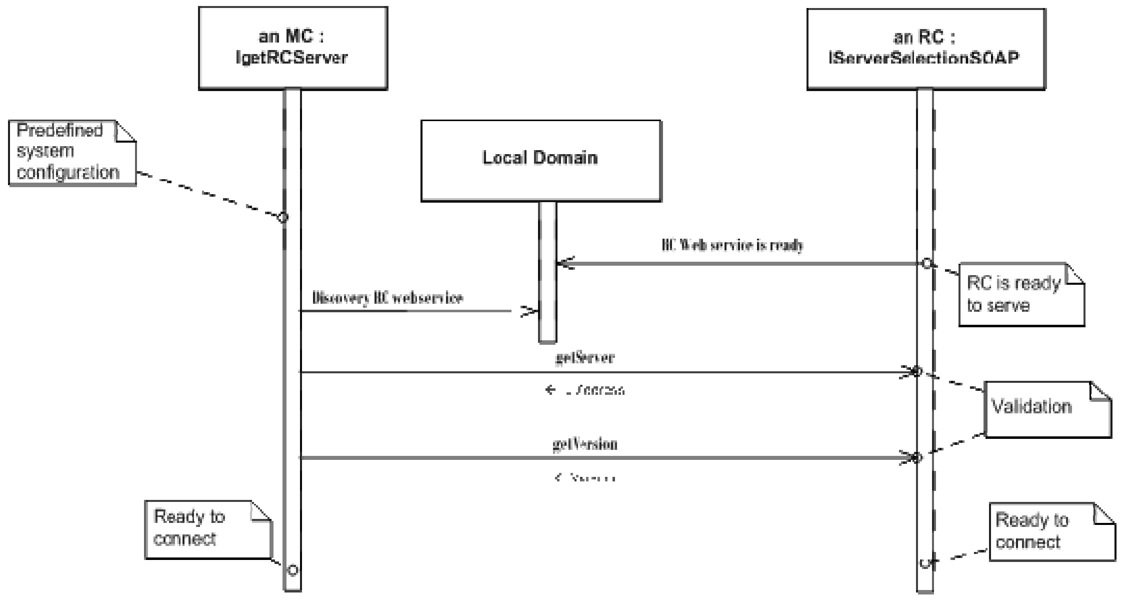


Figure 47 — Basic Flow Diagram for the Server Selection phase

8.3.1.2 Interface:IServerSelection

This operation will be performed using UDDI (Universal Description, Discovery and Integration) entries which completely describe the service providers and messages.

As a prerequisite, the MC shall get the full domain configuration from the Domain Controller server (Discovery RC web service) or from a predefined configuration table.

The following WSDL diagram describes the Web Services implementation of the IServerSelection Interface on the RC side.

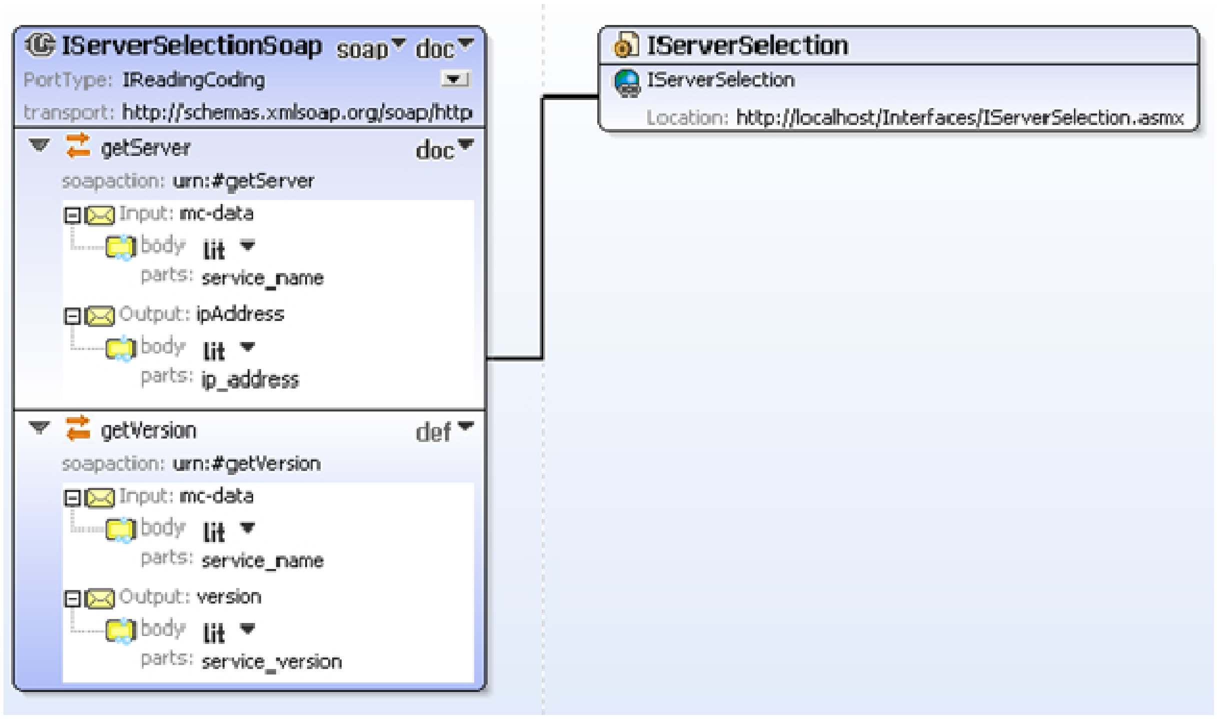


Figure 48

8.3.1.3 WSDL code

The following code is generated using XML Spy.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- edited with XMLSpy v2008 (http://www.altova.com) by () -->
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns="http://localhost/Interfaces/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:ns="http://localhost/SDD/" targetNamespace="http://localhost/Interfaces/">
    <binding name="IServerSelectionSoap" type="IReadingCoding">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="getServer">
            <soap:operation soapAction="urn:#getServer" style="document"/>
            <output name="ipAddress">
                <soap:body parts="ip_address" use="literal"/>
            </output>
            <input name="mc-data">
                <soap:body parts="service_name" use="literal"/>
            </input>
        </operation>
        <operation name="getVersion">
            <soap:operation soapAction="urn:#getVersion"/>
            <output name="version">
                <soap:body parts="service_version" use="literal"/>
            </output>
            <input name="mc-data">
                <soap:body parts="service_name" use="literal"/>
            </input>
        </operation>
    </binding>
    <service name="IServerSelection">
```

```
                <port name="IServerSelection" binding="IServerSelectionSoap">
                        <soap:address location="http://localhost/Interfaces/IServerSelection.asmx"/>
                </port>
        </service>
</definitions>
```

8.3.1.4 Exceptions Handling

The following exception handlers can be activated:

— Server Selection error

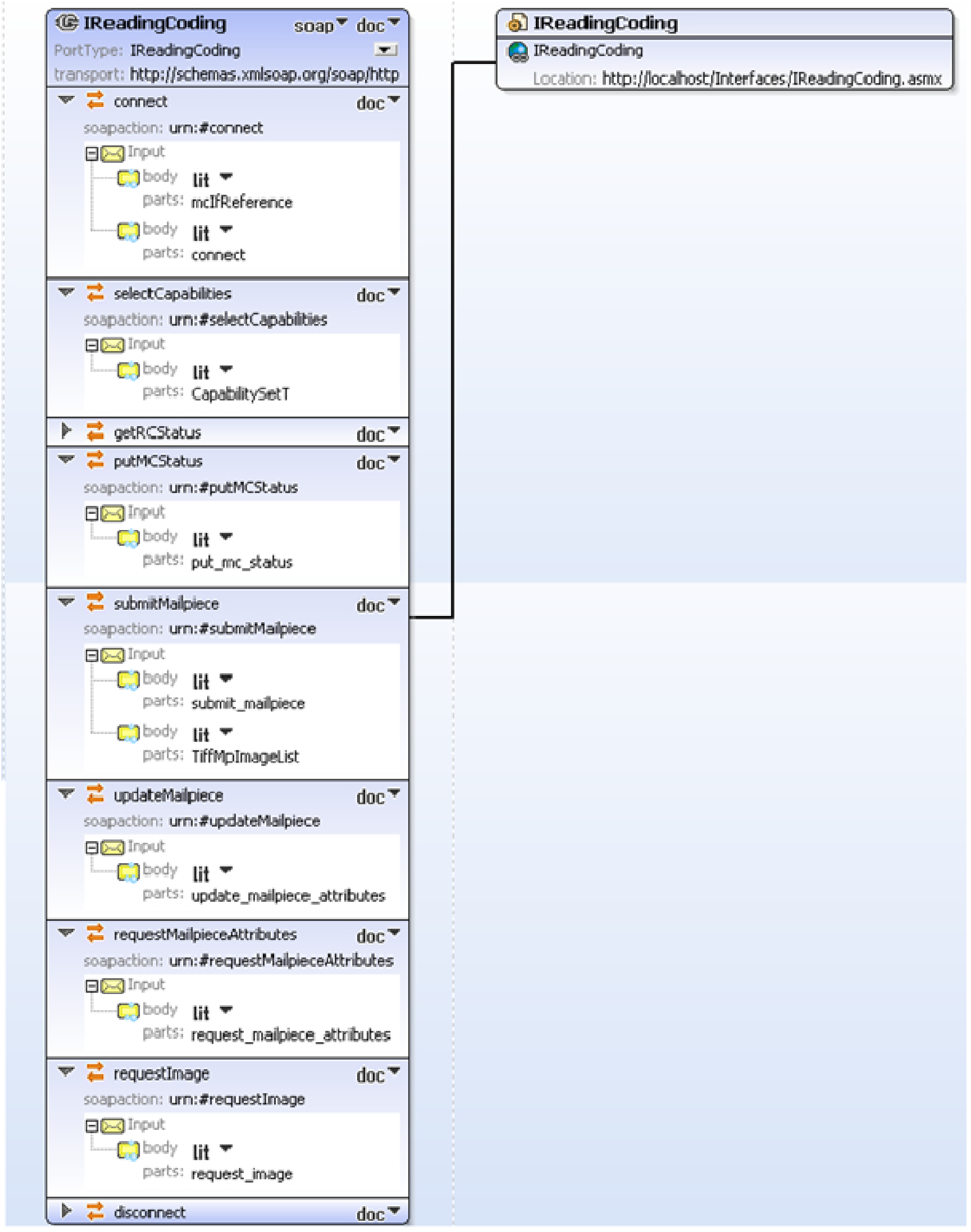
— IReadingCoding error

8.3.2 Interface Definition

In the following, WSDL is used for the concrete definition of the interfaces. Each service from the common part is mapped to a method of the same name in the WSDL. The data sent with each single web service operation, which is common to all implementations, is defined in the common part. Each interface is a Web Service that provides the described methods.

8.3.2.1 Interface: IReadingCoding

The following WSDL diagram describes Web Services implementation of the IReadingCoding Interface.


IReadingCoding soap doc
PortType: IReadingCoding
transport: http://schemas.xmlsoap.org/soap/http

connect doc
soapaction: urn:#connect
Input
body lit
parts: mcIfReference
body lit
parts: connect

selectCapabilities doc
soapaction: urn:#selectCapabilities
Input
body lit
parts: CapabilitySetT

getRCStatus doc

putMCStatus doc
soapaction: urn:#putMCStatus
Input
body lit
parts: put_mc_status

submitMailpiece doc
soapaction: urn:#submitMailpiece
Input
body lit
parts: submit_mailpiece
body lit
parts: TiffMpImageList

updateMailpiece doc
soapaction: urn:#updateMailpiece
Input
body lit
parts: update_mailpiece_attributes

requestMailpieceAttributes doc
soapaction: urn:#requestMailpieceAttributes
Input
body lit
parts: request_mailpiece_attributes

requestImage doc
soapaction: urn:#requestImage
Input
body lit
parts: request_image

disconnect doc

IReadingCoding
IReadingCoding
Location: http://localhost/Interfaces/IReadingCoding.asmx

Figure 49

8.3.2.1.1 WSDL code

The following code is generated using XML Spy.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- edited with XMLSpy v2008 (http://www.altova.com) by () -->
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns="http://localhost/Interfaces/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:ns="http://localhost/SDD/" targetNamespace="http://localhost/Interfaces/">
	<binding name="IReadingCoding" type="IReadingCoding">
		<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
		<operation name="connect">
			<soap:operation soapAction="urn:#connect" style="document"/>
			<input>
				<soap:body parts="mcIfReference" use="literal"/>
				<soap:body parts="connect" use="literal"/>
			</input>
		</operation>
		<operation name="selectCapabilities">
			<soap:operation soapAction="urn:#selectCapabilities" style="document"/>
			<input>
				<soap:body parts="CapabilitySetT" use="literal"/>
			</input>
		</operation>
		<operation name="getRCStatus">
			<soap:operation soapAction="urn:#getRCStatus" style="document"/>
		</operation>
		<operation name="putMCStatus">
			<soap:operation soapAction="urn:#putMCStatus" style="document"/>
			<input>
				<soap:body parts="put_mc_status" use="literal"/>
			</input>
		</operation>
		<operation name="submitMailpiece">
			<soap:operation soapAction="urn:#submitMailpiece" style="document"/>
			<input>
				<soap:body parts="submit_mailpiece" use="literal"/>
				<soap:body parts="TiffMpImageList" use="literal"/>
			</input>
		</operation>
		<operation name="updateMailpiece">
			<soap:operation soapAction="urn:#updateMailpiece" style="document"/>
			<input>
				<soap:body parts="update_mailpiece_attributes" use="literal"/>
			</input>
		</operation>
		<operation name="requestMailpieceAttributes">
			<soap:operation soapAction="urn:#requestMailpieceAttributes" style="document"/>
			<input>
				<soap:body parts="request_mailpiece_attributes" use="literal"/>
			</input>
		</operation>
		<operation name="requestImage">
			<soap:operation soapAction="urn:#requestImage" style="document"/>
			<input>
				<soap:body parts="request_image" use="literal"/>
			</input>
		</operation>
```

```
            <operation name="disconnect">
                  <soap:operation soapAction="urn:#disconnect" style="document"/>
            </operation>
      </binding>
      <service name="IReadingCoding">
            <port name="IReadingCoding" binding="IReadingCoding">
                  <soap:address location="http://localhost/Interfaces/IReadingCoding.asmx"/>
            </port>
      </service>
</definitions>
```

8.3.2.1.2 Exceptions Handling

The following exception handlers can be activated:

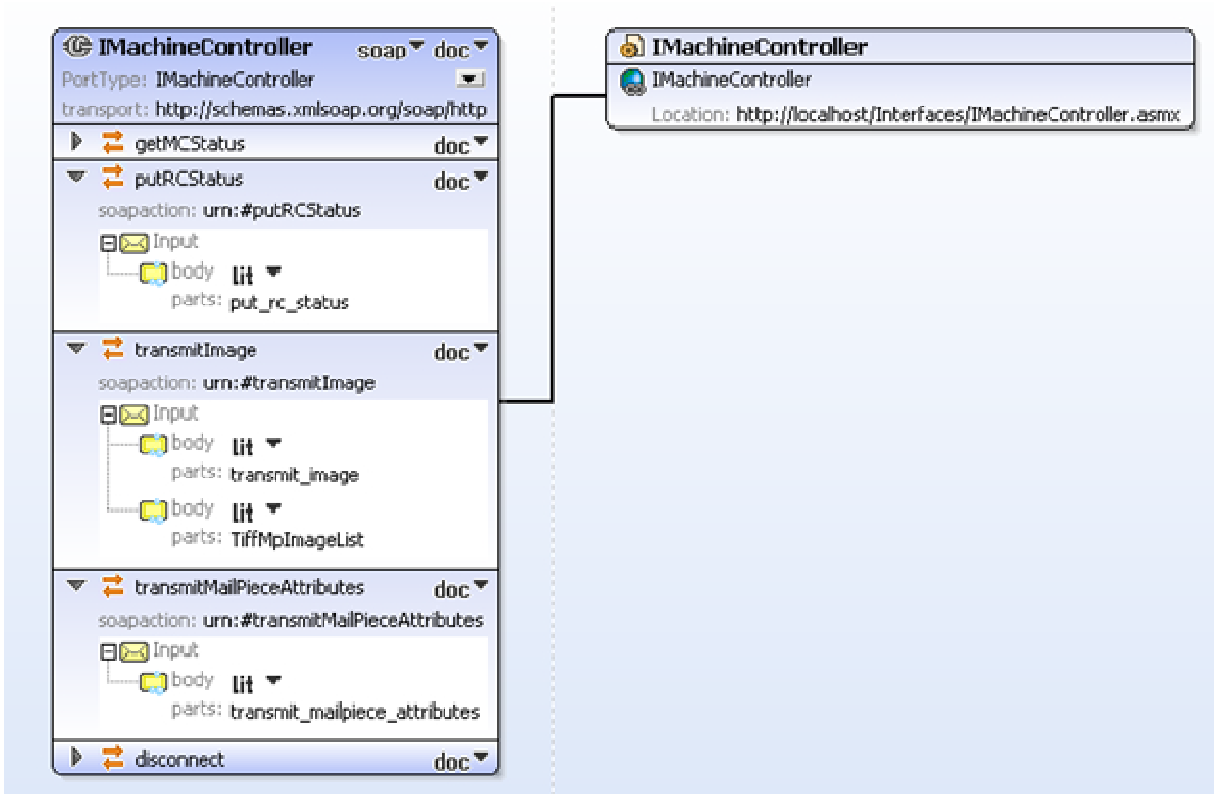
— Connection failed

— Not connected

— General error

8.3.2.2 Interface: IMachineController

The following WSDL diagram describes Web Services implementation of the IMachineController Interface.


```
IMachineController                         soap  doc
PortType: IMachineController
transport: http://schemas.xmlsoap.org/soap/http
getMCStatus                                      doc
putRCStatus                                      doc
  soapaction: urn:#putRCStatus
    Input
      body  lit
      parts: put_rc_status
transmitImage                                    doc
  soapaction: urn:#transmitImage
    Input
      body  lit
      parts: transmit_image
      body  lit
      parts: TiffMpImageList
transmitMailPieceAttributes                      doc
  soapaction: urn:#transmitMailPieceAttributes
    Input
      body  lit
      parts: transmit_mailpiece_attributes
disconnect                                       doc

IMachineController
IMachineController
  Location: http://localhost/Interfaces/IMachineController.asmx
```

Figure 50

8.3.2.2.1 WSDL code

The following code is generated using XML Spy.

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns="http://localhost/Interfaces/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:ns="http://localhost/SDD/" targetNamespace="http://localhost/Interfaces/">
	<binding name="IMachineController" type="IMachineController">
		<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
		<operation name="getMCStatus">
			<soap:operation soapAction="urn:#getMCStatus" style="document"/>
		</operation>
		<operation name="putRCStatus">
			<soap:operation soapAction="urn:#putRCStatus" style="document"/>
			<input>
				<soap:body parts="put_rc_status" use="literal"/>
			</input>
		</operation>
		<operation name="transmitImage">
			<soap:operation soapAction="urn:#transmitImage" style="document"/>
			<input>
				<soap:body parts="transmit_image" use="literal"/>
				<soap:body parts="TiffMpImageList" use="literal"/>
			</input>
		</operation>
		<operation name="transmitMailPieceAttributes">
			<soap:operation soapAction="urn:#transmitMailPieceAttributes" style="document"/>
			<input>
				<soap:body parts="transmit_mailpiece_attributes" use="literal"/>
			</input>
		</operation>
		<operation name="disconnect">
			<soap:operation soapAction="urn:#disconnect" style="document"/>
		</operation>
	</binding>
	<service name="IMachineController">
		<port name="IMachineController" binding="IMachineController">
			<soap:address location="http://localhost/Interfaces/IMachineController.asmx"/>
		</port>
	</service>
</definitions>
```

8.3.2.2.2 Exceptions Handling

The following exception handlers can be activated:

— General error

— Not connected

8.3.3 Used Types

The following types are used in the previous Interfaces.

service_name:

```
<types>
```

```
        <s:schema elementFormDefault="qualified" targetNamespace="http://localhost/Interfaces/">
                <s:element name="service_name">
                        <element name="name" type="string"/>
                </s:element>
        </s:schema>
</types>
```

ip_address:

```
<types>
        <s:schema elementFormDefault="qualified" targetNamespace="http://localhost/Interfaces/">
                <s:element name=" ipAddress ">
                        <element name="ip" type="string"/>
                </s:element>
        </s:schema>
</types>
```

service_version:

```
<types>
        <s:schema elementFormDefault="qualified" targetNamespace="http://localhost/Interfaces/">
                <s:element name="service_version">
                        <element name="version" type="string"/>
                </s:element>
        </s:schema>
</types>
```

mcIfReference:

```
<types>
        <s:schema elementFormDefault="qualified" targetNamespace="http://localhost/Interfaces/">
                <s:element name="TiffMpImage">
                                <s:complexType>
                                        <s:sequence>
                                                <s:element  name="service_identifier" type="service_name "/>
                                                <s:element  name="service_version_id" type="service_version "/>
                                        </s:sequence>
                                </s:complexType>
                        </s:element>
        </s:schema>
</types>
```

TiffMpImage, TiffMpImageList:

```
        <types>
                <s:schema elementFormDefault="qualified" targetNamespace="http://localhost/Interfaces/">
                        <s:element name="TiffMpImage">
                                <s:complexType>
                                        <s:sequence>
                                                <s:element minOccurs="1" name="chunck" type="s:byte"/>
                                        </s:sequence>
                                </s:complexType>
                        </s:element>
                        <s:element name="TiffMpImageList">
                <s:complexType>
                    <s:sequence>
                        <s:element minOccurs="1" name="Image" type="s:TiffMpImage"/>
                    </s:sequence>
                </s:complexType>
            </s:element>
        </s:schema>
    </types>
```

8.3.4 Exceptions Handling

Both the MC and the RC will support the exception handling in order to manage the anomalous conditions. Standard exceptions are generated following system level errors. Application dependent exceptions are managed with custom SoapException. Standardized error string definitions are in the IDD document.

Annex A
(informative)

XML data structure

A.1 Introduction to XML

This subclause is taken from CEN/TS 15448 and has not been changed. It is replicated here for the reader's convenience.

This subclause provides basic information regarding XML instance documents and XML schemas. It comprises some samples extracted from the domain specific schemas.

A.1.1 XML Document Structure

A well-formed XML instance document consists of three major parts:

— the Prologue – which represents an mandatory part in this specification;

— the Body – including the hierarchical element tree;

— and an optional Epilogue.

A.1.1.1 Prologue

The prologue signals the beginning of XML data. It describes the data's character encoding and comprises further configuration hints to the XML parser and the application. The prologue should provide at least the version identifier and the character encoding

So the minimal prologue should look as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

— **<?xml**	the string literal must be the very first characters of the document; no preceding whitespace or embedded comments are allowed.
— **Version**	this is required and its value currently has to be set to "1.0".
— **encoding**	It is recommended to use the "ISO-8859-x" encoding for country specific encoding. Most existing XML parsers support these encodings. UTF8 and UTF16 should be allowed. If the encoding is not specified in the XML instance document, UTF-8 encoding is assumed. Wrong encoding possibly leads to errors during parsing of the document. Normally an application has to terminate the parsing process; further processing of inconsistent input or output data should be suppressed.

A.1.1.2 Body

This is the main course of the XML data. The elements must comprise a hierarchical tree, with a single root node.

The XML data has to be valid. The requirement or term “valid” encloses, that the data ‘object’ should be well-formed, and that it meets certain validity constraints and matches further grammar describing the documents content. The description of the document structure is provided in the form of XML schemas.

A.1.1.3 Epilog

Not used yet.

A.2 Introduction to XML Schema

The W3C XML schema provides a way to constrain XML documents. It is a powerful and flexible language written in XML syntax. Therefore, it is not necessary to learn a different syntax for the creation of XML instance documents and XML schema specifications.

The following listing provides important features of schemas in general:

— Schemas offer an abstract description of the structure of documents an application will deal with (document exchange IC ↔ enrichment device).

— Schemas can provide a description how a document will be exchanged, which acts as an enforceable contract between the components.

— Schemas and the appropriate tool set relieve an application from the need to check or validate a XML document. It can pass it on to a COTS component, usually a XML parser.

— A schema is independent of any programming language type system. Therefore, it is predestined to be shared between communities (allows different companies to share the same model).

— Schemas are extensible; this bases on the ability of reusing parts of a schema in another schema or specifying types, which can be reused. A document instance can reference multiple schemas.

A.2.1 XML schema components

An XML schema is composed of the following four components:

— element declarations

— simple type definitions

— complex type definitions

— attribute and attribute group declarations

A.2.1.1 Elements versus Attributes

The domain data model prefers the use of elements for data items, which provide domain specific information that is in focus of content specific validation. This is to release an application from the burden of “out-of-the-box” validation. Elements in contrast to attributes allow a more general usage regarding object-oriented features like type derivation and inheritance. Attributes are unqualified, and can therefore not be globally defined (refer to [2], [3]).

A.2.1.1.1 Simple Types

Simple Types are atoms of information and cannot be split up. XML schema supports two varieties of data types, build-in types and user-derived types. In the domain data model the simple types represent the basic type specifications. The new types in the domain data model are all derived from build-in types.

EXAMPLE

```
<simpleType name="AngleRangeT">
    <restriction base="float">
        <minInclusive value="0.0"/>
        <maxInclusive value="360.0"/>
    </restriction>
</simpleType>
```

The type is derived from the build-in type *unsignedInt*. It limits the scale of values to the valid range of angles in a circuit.

A.2.1.1.2 Complex Types

Complex types enable a user to create element content models. It allows elements to carry attributes (in the form of element declarations).

EXAMPLE

```
<complexType name="DotType">
    <sequence>
        <element name="x" type="unsignedInt"/>
        <element name="y" type="unsignedInt"/>
    </sequence>
</complexType>
```

Content Models can be inherited and reused. Complex types can be assigned to element declarations.

EXAMPLE

```
<complexType name="QuadPolygonType">
    <sequence minOccurs="4" maxOccurs="4">
        <element name="dot" type="<namespace qualifier>:DotType"/>
    </sequence>
</complexType>
```

The example shows that complex types should be created as named types to enable reuse in the content model.

A.2.1.1.3 Attribute and Attribute Groups

Attributes and attribute groups are quite similar to the XML schema components elements and complex data types. Refer to [2], [3] for detailed information.

NOTE There are many more issues regarding the specification of XML schemas. Not addressed in this short introduction are:

— an overview about the build-in data types

— a detailed description of the content model

— aspects of data-types – value space, lexical space and facets

— the topics reuse, derivation, inheritance

— and all questions around the use and the specification of namespaces

Refer to [2], [3] to get a deeper understanding of XML and XML schemas. In particular [2] addresses the listed topics in an excellent way with many essential practical hints.

A.3 Listing of XML Schema

All schemas used for the definition of this interface have been listed and explained above. A schema definition file is also available and should always be distributed together with this document. However, for the reader's convenience in case no electronic schema is available, we here list the complete schema again.

```
<?xml version="1.0" encoding="utf-8"?>
<schema xmlns:mcrc="http://www.cen.com.au/Industry/2008/04/PostalAutomation/base" xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.cen.com.au/Industry/2008/04/PostalAutomation/base" elementFormDefault="qualified" version="0.1">
    <!-- ************************************************************ -->
    <!--    Start of MC-RC Domain Instance Types        -->
    <!-- ************************************************************ -->
    <!-- ****************************************** -->
    <!--    MC_RC_Message                    -->
    <!--    root tag for all messages        -->
    <!-- ****************************************** -->
    <element name="MC_RC_Message">
        <annotation>
            <documentation>common root tag for all messages on the interface</documentation>
        </annotation>
        <complexType>
            <choice>
                <element ref="mcrc:connect"/>
                <element ref="mcrc:disconnect"/>
                <element ref="mcrc:select_capabilities"/>
                <element ref="mcrc:get_rc_status"/>
                <element ref="mcrc:put_rc_status"/>
                <element ref="mcrc:get_mc_status"/>
                <element ref="mcrc:put_mc_status"/>
                <element ref="mcrc:submit_mailpiece"/>
                <element ref="mcrc:update_mailpiece_attributes"/>
                <element ref="mcrc:request_mailpiece_attributes"/>
                <element ref="mcrc:request_image"/>
                <element ref="mcrc:transmit_image"/>
                <element ref="mcrc:transmit_mailpiece_attributes"/>
            </choice>
        </complexType>
    </element>
    <!-- ****************************************** -->
    <!--    connect                          -->
    <!-- ****************************************** -->
    <element name="connect">
        <complexType>
            <sequence>
                <element name="interface_version_id" type="string"/>
                <element name="machine_controller_id" type="string"/>
                <element name="feeder_id" type="string"/>
                <element name="heartbeat_time_int" type="unsignedInt"/>
                <element name="initial_batch_information" type="mcrc:BatchInformationT" minOccurs="0"/>
            </sequence>
        </complexType>
    </element>
    <!-- ****************************************** -->
    <!--    disconnect                       -->
    <!-- ****************************************** -->
    <element name="disconnect"/>
    <!-- ****************************************** -->
    <!--    select_capabilities              -->
    <!-- ****************************************** -->
    <element name="select_capabilities" type="mcrc:CapabilitySetT"/>
    <!-- ****************************************** -->
    <!--    get_mc_status                    -->
    <!-- ****************************************** -->
    <element name="get_mc_status"/>
    <!-- ****************************************** -->
    <!--    put_mc_status                    -->
    <!-- ****************************************** -->
    <element name="put_mc_status">
        <complexType>
            <sequence>
                <element name="state" type="mcrc:MCStateT"/>
                <element name="text" type="string" minOccurs="0"/>
            </sequence>
        </complexType>
</element>
<!-- ******************************************* -->
<!--       get_rc_status                       -->
<!-- ******************************************* -->
<element name="get_rc_status"/>
<!-- ******************************************* -->
<!--       put_rc_status                        -->
<!-- ******************************************* -->
<element name="put_rc_status">
        <complexType>
                <sequence>
                        <element name="state" type="mcrc:RCStateT"/>
                        <element name="text" type="string" minOccurs="0"/>
                        <element name="selected_ready_capabilities" type="mcrc:CapabilitySetT"/>
                        <element name="selected_not_ready_capabilities" type="mcrc:CapabilitySetT"/>
                </sequence>
        </complexType>
</element>
<!-- ******************************************* -->
<!--       submit_mailpiece                     -->
<!-- ******************************************* -->
<element name="submit_mailpiece">
        <complexType>
                <complexContent>
                        <extension base="mcrc:MailpieceSubmissionT">
                                <attributeGroup ref="mcrc:MCRCInfrastructureDataT"/>
                                <attributeGroup ref="mcrc:ProcessingConfigurationT"/>
                                <attributeGroup ref="mcrc:ProcessingInstructionsT"/>
                        </extension>
                </complexContent>
        </complexType>
</element>
<!-- ******************************************* -->
<!--       update_mailpiece_attributes          -->
<!-- ******************************************* -->
<element name="update_mailpiece_attributes">
        <complexType>
                <sequence>
                        <element name="iood" type="boolean" minOccurs="0"/>
                        <element name="weight" type="unsignedInt" minOccurs="0"/>
                        <element name="indicia_value" type="string" minOccurs="0"/>
                        <element name="indicia_type" type="string" minOccurs="0"/>
                        <element name="sort_bin" type="unsignedInt" minOccurs="0"/>
                        <element name="idtag_print_status" type="boolean" minOccurs="0"/>
                        <element name="code" type="string" minOccurs="0"/>
                        <element name="final_coded" type="boolean" minOccurs="0"/>
                </sequence>
                <attribute name="mailpiece_id" type="string" use="required"/>
                <attribute name="source_id" type="string" use="optional"/>
                <attribute name="scanner_id" type="string" use="optional"/>
                <attribute name="feeder_id" type="string" use="optional"/>
                <attribute name="sortplan_id" type="string" use="optional"/>
        </complexType>
</element>
<!-- ******************************************* -->
<!--       request_mailpiece_attributes        -->
<!-- ******************************************* -->
<element name="request_mailpiece_attributes">
        <complexType>
                <attributeGroup ref="mcrc:MCRCInfrastructureDataT"/>
                <attributeGroup ref="mcrc:ProcessingInstructionsT"/>
        </complexType>
</element>
<!-- ******************************************* -->
<!--       request_image                      -->
<!-- ******************************************* -->
<element name="request_image" type="mcrc:ImageAttributesT"/>
<!-- ******************************************* -->
<!--       transmit_image                       -->
<!-- ******************************************* -->
<element name="transmit_image">
        <complexType>
                <sequence>
                <element name="request_parameters" type="mcrc:ImageAttributesT" minOccurs="0"/>
                <element name="image_status" type="mcrc:ImageStatusT" minOccurs="0"/>
                <element name="available_attributes" type="mcrc:RCResultT" minOccurs="0"/>
            </sequence>
        </complexType>
</element>
<!-- ****************************************** -->
<!--        transmit_mailpiece_attributes       -->
<!-- ****************************************** -->
<element name="transmit_mailpiece_attributes">
        <complexType>
            <complexContent>
                <extension base="mcrc:RCResultT">
                    <attributeGroup ref="mcrc:MCRCInfrastructureDataT"/>
                    <attributeGroup ref="mcrc:ProcessingInstructionsT"/>
                </extension>
            </complexContent>
        </complexType>
</element>
<!-- ********************************************************* -->
<!--         End MC-RC Domain Instance Types            -->
<!-- ********************************************************* -->
<!-- ********************************************************* -->
<!--            Start of MC-RC Generic Types               -->
<!-- ********************************************************* -->
<complexType name="RCResultT">
        <sequence>
            <element name="address_result" type="mcrc:AddressResultT" minOccurs="0"/>
            <element name="sort_bin" type="string" minOccurs="0"/>
            <element name="mp_size" type="mcrc:DimensionT" minOccurs="0"/>
            <element name="bar_code" type="mcrc:BarCodeT" minOccurs="0" maxOccurs="unbounded"/>
            <element name="print_text" type="mcrc:PrintTextT" minOccurs="0"/>
        </sequence>
</complexType>
<complexType name="AddressResultT">
        <sequence>
            <element name="code" type="string" minOccurs="0"/>
            <element name="code_attribute" type="string" minOccurs="0" maxOccurs="unbounded"/>
            <element name="final_coded" type="boolean" minOccurs="0"/>
            <element name="status_attr" type="string" minOccurs="0"/>
            <element name="result_source" type="string" minOccurs="0"/>
            <element name="reject_reason" type="string" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
</complexType>
<complexType name="PrintTextT">
        <sequence>
            <element name="type" type="string"/>
            <element name="text_line" type="string" maxOccurs="unbounded"/>
        </sequence>
</complexType>
<complexType name="CapabilitySetT">
        <sequence maxOccurs="unbounded">
            <element name="capability" type="mcrc:CapabilityT"/>
        </sequence>
</complexType>
<complexType name="ImageAttributesT">
        <sequence>
            <element name="feeder_id" type="string" minOccurs="0"/>
            <element name="scanner_id" type="string" minOccurs="0"/>
            <element name="source_id" type="string" minOccurs="0"/>
            <element name="type" type="mcrc:ImageTypeT"/>
            <element name="resolution" type="mcrc:ImageResolutionT"/>
        </sequence>
</complexType>
<complexType name="DimensionT">
        <sequence>
            <element name="long" type="unsignedInt" minOccurs="0"/>
            <element name="high" type="unsignedInt" minOccurs="0"/>
            <element name="thick" type="float" minOccurs="0"/>
        </sequence>
</complexType>
<complexType name="BarCodeT">
        <sequence>
            <element name="type" type="mcrc:BarcodeSymbologyT" minOccurs="0"/>
```

```
			<element name="value" type="string" minOccurs="0"/>
			<element name="status" type="mcrc:BarCodeStatusT"/>
		</sequence>
	</complexType>
	<complexType name="MailpieceSubmissionT">
		<sequence>
			<element name="task_set" type="mcrc:TaskT" minOccurs="0" maxOccurs="unbounded"/>
			<element name="mp_attr" type="mcrc:MailpieceAttrT" minOccurs="0"/>
		</sequence>
	</complexType>
	<complexType name="TaskT">
		<sequence>
			<element name="capability" type="mcrc:CapabilityT" minOccurs="0"/>
			<element name="batch_info" type="mcrc:BatchDescriptionT" minOccurs="0"/>
		</sequence>
	</complexType>
	<attributeGroup name="MCRCInfrastructureDataT">
		<attribute name="mailpiece_id" type="string" use="required"/>
		<attribute name="id_tag" type="string" use="optional"/>
	</attributeGroup>
	<complexType name="BatchInformationT">
		<sequence>
			<element name="batch_info" type="mcrc:BatchDescriptionT"/>
		</sequence>
		<attributeGroup ref="mcrc:ProcessingConfigurationT"/>
	</complexType>
	<!-- ************************************************************ -->
	<!--		End of MC-RC Generic Types		-->
	<!-- ************************************************************ -->
	<!-- ************************************************************ -->
	<!--		Start of MC-RC Primitive Types		-->
	<!-- ************************************************************ -->
	<simpleType name="MCStateT">
		<annotation>
			<documentation>
			- machine is running
			- machine is stopped
			- machine is in test mode
			- machine is in maintenance mode
			</documentation>
		</annotation>
		<restriction base="NMTOKEN">
			<enumeration value="running"/>
			<enumeration value="stopped"/>
			<enumeration value="test"/>
			<enumeration value="maintenance"/>
		</restriction>
	</simpleType>
	<simpleType name="RCStateT">
		<annotation>
			<documentation>
			- at least one selected capability is ready
			- a central component of the RC System is not available or none of the selected capabilities are ready
			- RC is connected but not yet fully started
			- RC is not connected
			</documentation>
		</annotation>
		<restriction base="NMTOKEN">
			<enumeration value="ready"/>
			<enumeration value="notready"/>
			<enumeration value="initializing"/>
			<enumeration value="notconnected"/>
		</restriction>
	</simpleType>
	<simpleType name="ImageTypeT">
		<annotation>
			<documentation>
				"all" means all available types
			</documentation>
		</annotation>
		<restriction base="NMTOKEN">
			<enumeration value="all"/>
			<enumeration value="binary"/>
			<enumeration value="grey"/>
```

```
                <enumeration value="color"/>
        </restriction>
</simpleType>
<simpleType name="ImageResolutionT">
        <annotation>
                <documentation>
                        "all" means all available resolutions
                </documentation>
        </annotation>
        <restriction base="NMTOKEN">
                <enumeration value="all"/>
                <enumeration value="high"/>
                <enumeration value="low"/>
        </restriction>
</simpleType>
<simpleType name="ImageStatusT">
        <annotation>
                <documentation>
                - image is available
                _image - no image with requested parameters is available
                _id - id of e.g feeder or scanner is unknown for the Machine
                </documentation>
        </annotation>
        <restriction base="NMTOKEN">
                <enumeration value="OK"/>
                <enumeration value="no_image"/>
                <enumeration value="invalid_id"/>
        </restriction>
</simpleType>
<simpleType name="BarCodeStatusT">
        <restriction base="NMTOKEN">
                <enumeration value="no_barcode"/>
                <enumeration value="detected_and_recognized"/>
                <enumeration value="detected_and_not_recognized"/>
        </restriction>
</simpleType>
<complexType name="MailpieceAttrT">
        <annotation>
                <documentation>
                type exists in OCR/VCS interface but without preknowledge.
                will contain the base IC-ED RecognitionResultT structure.
                </documentation>
        </annotation>
        <sequence>
                <element name="type" type="mcrc:MailpieceTypeT"/>
                <element name="mailclass" type="string" minOccurs="0"/>
                <element name="faces" type="mcrc:FaceSetT" minOccurs="0"/>
                <element name="preknowledge" type="mcrc:RecognitionResultT" minOccurs="0"/>
        </sequence>
</complexType>
<simpleType name="ActionT">
        <annotation>
                <documentation>
                - recognition is expected online
                - recognition is expected offline
                - recognition is expected both on- and offline
                </documentation>
        </annotation>
        <restriction base="NMTOKEN">
                <enumeration value="online"/>
                <enumeration value="offline"/>
                <enumeration value="mixed"/>
        </restriction>
</simpleType>
<!-- ************************************************************ -->
<!--            End of MC-RC Primitive Types                      -->
<!-- ************************************************************ -->
<!-- ********************************************************************************************************* -->
<!--            All Types below this line origin from the CEN OCR/VCS Interface            -->
<!--            There have been no changes to the types except the namespace          -->
<!-- ********************************************************************************************************* -->
<!-- ************************************************************ -->
<!--            Start of CEN OCR/VCS Generic Types    -->
<!-- ************************************************************ -->
```

```
	<complexType name="CapabilityT">
		<sequence>
			<element name="type" type="mcrc:CapabilityTypeT"/>
			<element name="sub_type" type="string" minOccurs="0" maxOccurs="unbounded"/>
			<element name="symbology" type="string" minOccurs="0" maxOccurs="unbounded"/>
			<element name="action" type="mcrc:ActionT" minOccurs="0" maxOccurs="unbounded"/>
		</sequence>
	</complexType>
	<attributeGroup name="CreatorT">
		<attribute name="creator" type="string" use="optional"/>
		<attribute name="version" type="string" use="optional"/>
		<attribute name="OD" type="string" use="optional"/>
	</attributeGroup>
	<attributeGroup name="ProcessingInstructionsT">
		<attribute name="permitted_time" type="unsignedInt" use="optional"/>
	</attributeGroup>
	<attributeGroup name="ProcessingConfigurationT">
		<attribute name="source_id" type="string" use="optional"/>
		<attribute name="feeder_id" type="string" use="optional"/>
		<attribute name="sortplan_id" type="string" use="optional"/>
		<attribute name="lift_location" type="string" use="optional"/>
		<attribute name="geo_mode" type="mcrc:BatchCntryCodeAttrT" use="optional"/>
		<attribute name="mail_flow" type="string" use="optional"/>
	</attributeGroup>
	<complexType name="RecognitionResultT">
		<sequence>
			<element name="code" type="string" minOccurs="0"/>
			<element name="code_cnf" type="mcrc:ConfidenceRangeT" minOccurs="0"/>
			<element name="code_attr" type="string" minOccurs="0" maxOccurs="unbounded"/>
			<element name="status_attr" type="string" minOccurs="0" maxOccurs="unbounded"/>
			<element name="text" type="string" minOccurs="0"/>
			<element name="reject" type="string" minOccurs="0"/>
			<element name="region" type="mcrc:RegionT" minOccurs="0"/>
			<element name="recognized_elements" type="mcrc:TextElementT" minOccurs="0" maxOccurs="unbounded"/>
			<element name="standardized_elements" type="mcrc:StandardizedElementT" minOccurs="0"
maxOccurs="unbounded"/>
			<element name="location" type="mcrc:LocationT" minOccurs="0"/>
			<element name="attr_dict" type="mcrc:AttributeDictionaryT" minOccurs="0"/>
		</sequence>
		<attributeGroup ref="mcrc:CreatorT"/>
	</complexType>
	<complexType name="TextElementT">
		<sequence>
			<element name="type" type="string"/>
			<element name="nominal" type="mcrc:NominalT" minOccurs="0"/>
			<element name="reference" type="mcrc:ReferenceT" minOccurs="0" maxOccurs="unbounded"/>
		</sequence>
	</complexType>
	<complexType name="NominalT">
		<sequence>
			<element name="type" type="string"/>
			<element name="code" type="string"/>
			<element name="code_cnf" type="mcrc:ConfidenceRangeT"/>
			<element name="attribute" type="string" minOccurs="0" maxOccurs="unbounded"/>
			<element name="location" type="mcrc:LocationT" minOccurs="0"/>
		</sequence>
	</complexType>
	<complexType name="ReferenceT">
		<sequence>
			<element name="type" type="string"/>
			<element name="code" type="string"/>
			<element name="match_cnf" type="mcrc:ConfidenceRangeT" minOccurs="0"/>
			<element name="attribute" type="string" minOccurs="0" maxOccurs="unbounded"/>
		</sequence>
	</complexType>
	<complexType name="StandardizedElementT">
		<sequence>
			<element name="type" type="string"/>
			<element name="code" type="string"/>
			<element name="attribute" type="string" minOccurs="0" maxOccurs="unbounded"/>
		</sequence>
	</complexType>
	<complexType name="RegionT">
		<sequence>
```

```
                    <element name="type" type="mcrc:InfoCarrierT"/>
                    <element name="symbology" type="string" minOccurs="0"/>
            </sequence>
    </complexType>
    <!-- ********************************************************** -->
    <!--              End of CEN OCR/VCS Generic Types     -->
    <!-- ********************************************************** -->
    <!-- ********************************************************** -->
    <!--               Start of CEN OCR/VCS Base Types      -->
    <!-- ********************************************************** -->
    <complexType name="BatchDescriptionT" mixed="false">
            <annotation>
                    <documentation>
    Batch characteristics section
      country ID element is valid only if code_attr = export.  Only a single country ID allowed.
  </documentation>
            </annotation>
            <sequence>
                    <element name="mailer_id" type="string" minOccurs="0"/>
                    <element name="batch_id" type="string" minOccurs="0"/>
                    <element name="code_attr" type="mcrc:BatchCodeAttrT" minOccurs="0" maxOccurs="unbounded"/>
                    <element name="status_attr" type="mcrc:BatchStatusAttrT" minOccurs="0"/>
                    <element name="geo_mode" type="mcrc:BatchCntryCodeAttrT" minOccurs="0"/>
                    <element name="cntry_id" type="string" minOccurs="0"/>
                    <element name="region" type="mcrc:AddressRegionT" minOccurs="0"/>
                    <element name="location" type="mcrc:LocationT" minOccurs="0"/>
                    <element name="restrictions" type="mcrc:RestrictionT" minOccurs="0" maxOccurs="unbounded"/>
                    <element name="mail_flow" type="string" minOccurs="0"/>
            </sequence>
            <attributeGroup ref="mcrc:CreatorT"/>
    </complexType>
    <simpleType name="BatchCodeAttrT">
            <annotation>
                    <documentation>
                    envelope characteristics
                        env_var         type of envelop varies
                        env_normal   type of envelop is normal
            _plastic      type of envelop is plastic
                    print quality
                        pqua_std                print quality of address is standard
            _low          low-contrast print quality of address
            _colour      coloured print of address
                    background characteristics
            _text          textured background in address block area
                        bgr_unln                background in address block area with underlines
                    info about the font type
            _var           font type varies
                        fntt_std          standard font type
            _italic                 italic font
                        fntt_dmatrix   dot-matrix font
                    font pitch characteristics
            _var           font pitch varies
                        fntp_prop                proportinal font pitch
            _const                constant font pitch
            </documentation>
            </annotation>
            <restriction base="NMTOKEN">
                    <enumeration value="env_unk"/>
                    <enumeration value="env_normal"/>
                    <enumeration value="env_plastic"/>
                    <enumeration value="pqua_std"/>
                    <enumeration value="pqua_low"/>
                    <enumeration value="pqua_colour"/>
                    <enumeration value="bgr_std"/>
                    <enumeration value="bgr_dark"/>
                    <enumeration value="bgr_colour"/>
                    <enumeration value="bgr_text"/>
                    <enumeration value="bgr_unln"/>
                    <enumeration value="fntt_var"/>
                    <enumeration value="fntt_std"/>
                    <enumeration value="fntt_italic"/>
                    <enumeration value="fntt_dmatrix"/>
                    <enumeration value="fntp_var"/>
                    <enumeration value="fntp_prop"/>
```

```
                <enumeration value="fntp_const"/>
            </restriction>
    </simpleType>
    <simpleType name="BatchStatusAttrT">
            <annotation>
                <documentation>
            _unk origin of mail in the batch is unknown
            _sm  origin of mail in the batch is single mailer
            _mm         origin of mail in the batch is multiple mailer
            </documentation>
            </annotation>
            <restriction base="NMTOKEN">
                <enumeration value="origin_unk"/>
                <enumeration value="origin_sm"/>
                <enumeration value="origin_mm"/>
            </restriction>
    </simpleType>
    <complexType name="AddressRegionT" mixed="false">
            <annotation>
                <documentation> specific region types </documentation>
            </annotation>
            <complexContent mixed="false">
                <restriction base="mcrc:RegionT">
                    <sequence>
                        <element name="type" type="mcrc:InfoCarrierT" fixed="address"/>
                        <element name="symbology" type="mcrc:TextSymbologyT"/>
                    </sequence>
                </restriction>
            </complexContent>
    </complexType>
    <simpleType name="BatchCntryCodeAttrT">
            <annotation>
                <documentation>
    geographic mode characteristics: specifies the geographic sorting mode of the machine. The values can be:
      national - only national domestic mail is sorted
      import - mail coming only from foreign countries to the local country
      export - mail to be delivered only to foreign countries
      mix - all kinds of mail
</documentation>
            </annotation>
            <restriction base="NMTOKEN">
                <enumeration value="national"/>
                <enumeration value="import"/>
                <enumeration value="export"/>
                <enumeration value="mix"/>
            </restriction>
    </simpleType>
    <!-- ********************************************************** -->
    <!--              End of CEN OCR/VCS Base Types        -->
    <!-- ********************************************************** -->
    <!-- ********************************************************** -->
    <!--            Start of CEN OCR/VCS Primitive Types    -->
    <!-- ********************************************************** -->
    <simpleType name="CapabilityTypeT">
            <annotation>
                <documentation>
                type contains simple domain-specific capabilities
                _code – recipient address recognition
                _code – sender address recognition
                _code – mailclass determination
                _code – endorsement line recognition
                _code – distribution barcode recognition
                _code – customer barcode recognition
                _code – product barcode recognition
                _management – Storage/Query of enrichment results e.g. by ID Tag
                _request – request image by MC, e.g. for selection of batch region
                _result_trans – Synchronous result transmission
                _result_trans – Asynchronous result transmission
                </documentation>
            </annotation>
            <restriction base="NMTOKEN">
                <enumeration value="rec_code"/>
                <enumeration value="sen_code"/>
                <enumeration value="mcl_code"/>
            <enumeration value="end_code"/>
            <enumeration value="dbc_code"/>
            <enumeration value="cbc_code"/>
            <enumeration value="pbc_code"/>
            <enumeration value="result_management"/>
            <enumeration value="image_request"/>
            <enumeration value="synch_result_trans"/>
            <enumeration value="asynch_result_trans"/>
        </restriction>
    </simpleType>
    <simpleType name="MailpieceFaceT">
        <restriction base="NMTOKEN">
            <enumeration value="top"/>
            <enumeration value="bottom"/>
            <enumeration value="left"/>
            <enumeration value="right"/>
            <enumeration value="front"/>
            <enumeration value="rear"/>
        </restriction>
    </simpleType>
    <simpleType name="AngleRangeT">
        <restriction base="float">
            <minInclusive value="0.0"/>
            <maxInclusive value="360.0"/>
        </restriction>
    </simpleType>
    <complexType name="RestrictionT">
        <annotation>
            <documentation> complex domain specific types </documentation>
        </annotation>
        <sequence>
            <element name="type" type="mcrc:RestrictionTypeT"/>
            <element name="location" type="mcrc:LocationT"/>
        </sequence>
    </complexType>
    <simpleType name="RestrictionTypeT">
        <restriction base="NMTOKEN">
            <enumeration value="include"/>
            <enumeration value="exclude"/>
        </restriction>
    </simpleType>
    <simpleType name="MailpieceTypeT">
        <restriction base="NMTOKEN">
            <enumeration value="unknown"/>
            <enumeration value="letter"/>
            <enumeration value="letter_bundle"/>
            <enumeration value="flat"/>
            <enumeration value="flat_bundle"/>
            <enumeration value="parcel"/>
        </restriction>
    </simpleType>
    <complexType name="FaceSetT">
        <sequence>
            <element name="face" maxOccurs="unbounded">
                <complexType>
                    <simpleContent>
                        <extension base="mcrc:MailpieceFaceT">
                            <attribute name="page" type="unsignedInt" use="optional"/>
                        </extension>
                    </simpleContent>
                </complexType>
            </element>
        </sequence>
    </complexType>
    <simpleType name="BarcodeSymbologyT">
        <annotation>
            <documentation>
            USPS Postnet barcode
            USPS Planet barcode
        SCC Royal Mail 4-State Customer Code
        SC  USPS 4-State Code
                Netherlands 4-State Customer Code
                Australian Postal Code - 4-State Barcode
            of5         2 of 5 Code - Industrial
        of5         Interleaved 2 of 5 Code

                    Code 39
                    Code 93
                Code 128
        A
        E           short version of UPC-A
                    European Article Number - Length 8


                Reduced Space Symbology
                Potable Data File 417



        </documentation>
        </annotation>
        <restriction base="NMTOKEN">
                <enumeration value="Postnet"/>
                <enumeration value="Planet"/>
                <enumeration value="RM4SCC"/>
                <enumeration value="USPS4SC"/>
                <enumeration value="KIXC"/>
                <enumeration value="AUPC"/>
                <enumeration value="2of5"/>
                <enumeration value="I2of5"/>
                <enumeration value="Codabar"/>
                <enumeration value="AIM39"/>
                <enumeration value="AIM93"/>
                <enumeration value="AIM128"/>
                <enumeration value="UPC-A"/>
                <enumeration value="UPC-E"/>
                <enumeration value="EAN-8"/>
                <enumeration value="EAN-13"/>
                <enumeration value="EAN-128"/>
                <enumeration value="RSS-14"/>
                <enumeration value="PDF417"/>
                <enumeration value="Aztec"/>
                <enumeration value="DataMatrix"/>
                <enumeration value="MaxiCode"/>
                <enumeration value="FIM"/>
        </restriction>
</simpleType>
<simpleType name="InfoCarrierT">
        <annotation>
                <documentation>
  address - Address Block Region
  barcode - Barcode Region
  logo     - Region of a Logo
  endln    - Endorsement Line Region
 </documentation>
        </annotation>
        <restriction base="NMTOKEN">
                <enumeration value="address"/>
                <enumeration value="barcode"/>
                <enumeration value="logo"/>
                <enumeration value="endln"/>
        </restriction>
</simpleType>
<simpleType name="TextSymbologyT">
        <annotation>
                <documentation> simple symbology types
         printed text - imprint
                 written text
        </documentation>
        </annotation>
        <restriction base="NMTOKEN">
                <enumeration value="machine"/>
                <enumeration value="script"/>
        </restriction>
</simpleType>
<complexType name="LocationT">
        <sequence>
                <element name="face" type="mcrc:MailpieceFaceT" minOccurs="0"/>
```

```
            <element name="angle" type="mcrc:AngleRangeT" minOccurs="0"/>
            <element name="confidence" type="mcrc:ConfidenceRangeT" minOccurs="0"/>
            <element name="polygon" type="mcrc:PolygonT"/>
        </sequence>
        <attribute name="page" type="unsignedInt" use="optional"/>
    </complexType>
    <simpleType name="ConfidenceRangeT">
        <restriction base="float">
            <minInclusive value="0.0"/>
            <maxInclusive value="1.0"/>
        </restriction>
    </simpleType>
    <complexType name="PolygonT">
        <sequence>
            <element name="dot" minOccurs="4" maxOccurs="unbounded">
                <complexType>
                    <attribute name="x" type="unsignedInt" use="required"/>
                    <attribute name="y" type="unsignedInt" use="required"/>
                </complexType>
            </element>
        </sequence>
    </complexType>
    <complexType name="AttributeT">
        <sequence>
            <element name="name" type="string"/>
            <element name="val" type="string"/>
        </sequence>
    </complexType>
    <complexType name="AttributeDictionaryT">
        <sequence>
            <element name="id" type="string"/>
            <element name="attr" type="mcrc:AttributeT" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <!-- *********************************************************** -->
    <!--            End of CEN OCR/VCS Primitive Types   -->
    <!-- *********************************************************** -->
</schema>
```

Bibliography

[1] UML 2.2 specification (formal / 2009-02-04), OMG, http://www.omg.org/spec/UML/2.2/

[2] Eric van der Vliest: XML Schemas, O'Reilly, ISBN: 0.596-00252-1

[3] Wrox: Professional XML – 2nd edition

[4] ISO/IEC 8802-3, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*

[5] ANSI/IEEE 802.3, *Standard for Information Technology – Telecommunications and Information Exchange Between Systems – Local and Metropolitan Area Networks – Specific Requirements – Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and P*

[6] IEEE 802.1x, *IEEE Standard for Local and metropolitan area networks – Port-Based Network Access Control*

[7] IEEE 802.2, *Logical Link Control*

[8] RFC 791, *Internet Protocol*

[9] RFC 792, *Internet Control Message Protocol*

[10] RFC 793, *Transmission Control Protocol*

[11] RFC 826, *Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*

[12] RFC 894, *A Standard for the Transmission of IP Datagrams over Ethernet Networks*

[13] UPU Standards glossary[1)]

[1)] UPU Standards are obtainable from the UPU International Bureau, whose contact details are given in the Bibliography; the UPU Standards glossary is freely accessible on URL http://www.upu.int.

This page deliberately left blank

This page deliberately left blank